

WarpOS

Sam Jordan

COLLABORATORS

	<i>TITLE :</i> WarpOS		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Sam Jordan	July 19, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	WarpOS	1
1.1	WarpOS	1
1.2	Preface	1
1.3	Introduction	2
1.4	A first overview	3
1.5	Multitasking	4
1.6	The WarpOS-Kernel	5
1.7	The Task-structure	6
1.8	The Task-functions	9
1.9	Context Switches	11
1.10	Signaling	12
1.11	Message-Handling	13
1.12	Memory Management	15
1.13	Semaphores	17
1.14	Lists / Tag Lists	17
1.15	Hardware Interfaces	18
1.16	Exception Handling	21
1.17	WarpOS-Debugger	23
1.18	Preferences	23
1.19	Tool-Programs	25
1.20	Demo-Programs	29
1.21	Developer Support	35
1.22	Frequently Asked Questions	37
1.23	Thanks	37
1.24	Literature Index	38
1.25	Support	38

Chapter 1

WarpOS

1.1 WarpOS

powerpc.library V14 / WarpUP-WarpOS

1998 by Sam Jordan
© HAAGE & PARTNER Computer GmbH

The PowerPC-Operating system for Warp-Speed

'Welcome to warp speed!'

Preface	Hardware Interfaces
Introduction	Exception Handling
A first overview	WarpOS-Debugger
Context Changes	Preferences
Multitasking	Tool-Programs
Signaling	Demo-Programs
Message-Handling	Developer Support
Memory Management	Thanks
Semaphores	Literature Index
Lists / Taglists	Support

1.2 Preface

It all began at the end of November 1996.

That was when I got my PPC-Board. My PowerASM-assembler had progressed far enough to use it for creating PPC-programs. One program ('cybermand') had already been created and extended with PowerPC-support. The climax was scheduled for one week later: 'cybermand' was supposed to show what the PowerPC-processor could really do.

What was designed as a highlight became a disaster. 'Cybermand' worked... a lot slower on the PPC than on the 68060. My disappointment soon changed into

outrageous anger because the reason for this performance-disaster lay in the communications-interface supplied with the PPC-board. Only one day later we at HAAGE&PARTNER decided to create our communications-interface for the PowerPC because we were not at all willing to live with such an abysmal performance.

Another week later I first saw a PowerAMIGA at work.

Thus began the development of the powerpc.library - a library, which has changed considerably as time progressed. The development can be broken down into three stages:

- V1-6: direct hardware-access, not multitasking-capable, depended upon the ppc.library when starting, very high performance
- V7: hardware-independent, based on the ppc.library, very weak performance (because of the ppc.library, of course)
- V8+: PowerPC-operating system WarpOS, completely independent of the ppc.library, highest possible performance

The step from v6 to V7 might look a bit strange, but politics were the cause for this step, not technical reasons.

WarpOS was developed with the goal in mind to optimally integrate the PowerPC into the current AMIGA-OS and achieve maximum performance for the necessary context switches - a goal which has most definitely been achieved. The best example for this is most likely the 'voxelspace' demo-program which quite impressively shows what the PowerAMIGA can do if the right interface-software is installed.

At the same time, WarpOS is most likely the only chance to create super-fast games for the PowerPC. Hints for optimal games-programming can be found in the 'GameDev.guide' document.

A primary goal of HAAGE&PARTNER and me will always be to do our part to keep the AMIGA rolling into a better future. I wish all users of WarpOS lots of fun with their PowerAMIGA!

Sam Jordan

1.3 Introduction

WarpOS is the first PowerPC-operating system featuring native multitasking for all AMIGAs that have both a 68K- and a PPC-processor. WarpOS is roughly comparable to 'exec', the operating system-kernel of AMIGA-OS. For this reason it comes as no surprise that many functions of WarpOS are mirror images of their respective exec-counterparts.

WarpOS is directly tied to the powerpc.library. WarpOS is integrated into the powerpc.library and is booted as soon as the powerpc.library is opened for the first time.

The powerpc.library is a mixed shared library that contains functions for both the 68K and PPC-processor.

Very important: As soon as WarpOS has been booted, absolutely NO PPC-programs

that are present in ELF-format will run. If compatibility with ELF-programs is desired, the powerpc.library version 7 must be installed - after that, ELF-programs as well as programs using the extended hunk-format will run. The price you have to pay for this is a miserable performance when context-switches between the CPUs occur. Furthermore, most of the features of WarpOS will not be usable (version 7 is not an operating system in its own right).

Below you find a list of the most important features of WarpOS:

- high-speed communication-interface between the 68K- and PPC-CPU
- completely native multitasking. Since V14 a new, very powerful dynamic scheduler is implemented.
- memory management, semaphores, lists/tag-management, signalling, message handling - all entirely native
- facultative memory protection: task are given the possibility to allocate protected memory
- virtual signals, this means signals are CPU-shared and are always re-directed to the correct CPU
- Inter-CPU message-system: messages can be passed between the CPUs
- optimal use of the PPC-MMU and the PPC-Cache
- MMU/Exception handling-support for applications
- powersave-function if no PPC-applications are running
- PowerPC-enforcer (protection of the first page)
- detailed crash-requester that optimally supports the developer when tracking bugs
- integrated debugging-system to easily locate errors
- special support for highly optimized software such as games/demos
- comprehensive developer documentation for optimal development of PPC-software

WarpOS and most of the additional programs were developed in assembler - the biggest part using the StormPowerASM (PPC-Assembler).

1.4 A first overview

This chapter should present you with an overview over the entire system. The core of the operating system are the three shared-libraries 'powerpc.library', 'warp.library' and 'warphw.library'

The 'powerpc.library' is the actual heart of WarpOS as WarpOS is integrated into the powerpc.library. All features of WarpOS can be used through calls to various functions of the powerpc.library which contains functions for the 68K as well as the PPC.

The 'warp.library' is the hardware-interface to the PowerPC. It offers a lot of functions that operate very close to the hardware. This library is used by WarpOS and is not documented because applications should always use the powerpc.library if access to the hardware is desired.

The 'warphw.library' is the hardware driver for the WarpUp-HAL. In the directory 'hwdrivers' there are detailed documentation and examples sources which allow everyone to write WarpUp drivers for new PPC hardware. For every hardware there exists a library 'warphw.library'. It is absolutely necessary that the correct library is installed. This can be tested by

running the tool 'GetDriverInfo' (in the 'tools' directory) which prints out, which hardware is represented by this driver.

This archive also contains all developer material needed to develop PPC-software. This material consists of:

- Include-files (assembler):

```
powerpc/ppcmacros.i
powerpc/powerpc.i
powerpc/listsPPC.i
powerpc/memoryPPC.i
powerpc/tasksPPC.i
powerpc/semaphoresPPC.i
powerpc/portsPPC.i
libraries/powerpc.i
```

- Include-files (C)

```
clib/powerpc_protos.h
stormprotos/powerpc_sprotos.h
pragma/powerpc_lib.h
libraries/powerpc.h
powerpc/powerpc.h
powerpc/memoryPPC.h
powerpc/tasksPPC.h
powerpc/semaphoresPPC.h
powerpc/portsPPC.h
```

- LVO-files that contain all library-offsets (assembler):

```
powerpc_lib.i
```

- documentation of the library-functions of the powerpc.library

```
powerpc.doc
powerpc.guide
```

In addition, the archive contains a number of source codes for learning purposes. For a closer look at the developer material take a look at the Developer Support chapter.

On top of the developer material there are also a few tool- and demo-programs contained that either fulfill useful roles or demonstrate the capabilities of WarpOS. These are covered more extensively in the Tool-programs and Demo-programs chapters.

1.5 Multitasking

This chapter gives a detailed description of WarpOS' multitasking as well as a description of the necessary structures and functions for task-handling.

The WarpOS-Kernel
The Task-Structure
The Task-Functions

1.6 The WarpOS-Kernel

In close similarity to exec, the core of WarpOS is formed by the scheduler which handles all task-priorities as well as switching between tasks. The WarpOS-scheduler is implemented as a standard exception-handler for the decrementer-interrupt.

One of the biggest difference between older versions and newer versions from V14 on is the new WarpOS scheduler. It works completely different than the old scheduler and therefore also completely different to exec's scheduler. For information related to the old scheduler refer to the documentation in older releases of WarpUp. Now the new scheduler is explained more detailed.

Please note that all these features which will be mentioned are only valid for the PowerPC side since exec is not a dynamic scheduler. Full dynamic scheduling can be achieved using the software 'Executive' (can be found on Aminet).

An overview of the features of the new scheduler:

- The new scheduler doesn't support fixed priorities anymore, the distribution of CPU time is now done dynamically. This means: no blocking due to running tasks with high priority anymore!
 - Tasks which are only active for a short time get a higher priority and more CPU time for a short time. This means that interactions of the user are handled much faster, for example, new tasks are launched very quickly, even if the system is massively loaded.
 - The CPU time each task gets is derived from its activity, this means, that a task with about 80 percent activity gets more time as a task with about 40 percent activity.
 - The distribution of CPU time is regulated and observed internally. The scheduler tries to keep the multitasking as constant as possible over a defined period of time.
 - If the system is very loaded the tasks get a smaller time slot, so that they get the CPU more often.
 - The scheduler supports the so-called NICE values which can be seen as a sort of replacement of the old priorities. These values allow to affect the CPU time of each task in a flexible way. As soon as several tasks run permanently, it's even possible to set the execution speed of the task!
 - The dynamic scheduler creates statistics for each task and for the whole system. This makes it possible to create such popular tools like 'CPUMeter'. In this archive a shell-based tool ('stat', explained in the chapter
-

Tool-Programs) can be found which displays these statistics.

- WarpOS now also supports ID numbers for tasks which ease the handling of the tasks, especially the removing of crashed tasks using the tool 'killppc' (explained in the chapter Tool-Programs).

The user has different possibilities to affect the scheduling, the most important one is the distribution of NICE values.

The NICE values give information about how nice a task is to other tasks. The smaller the value the less nice it is and the more CPU time it requests. For higher values the opposite occurs and the task leaves more CPU time to the other tasks. The NICE values can be in range of -20 to 20. These values can be set either when creating new tasks using CreateTaskPPC() or using the tool 'niceppc' (explained in the chapter Tool-Programs).

The tool 'niceppc' uses the new function 'SetNiceValue' which expects the task address and the NICE value. In this way the NICE value of all tasks can be affected also from other applications.

Another possibility to affect the scheduler is the tool program 'sched' which defines how much low-activity-tasks are preferred against high-activity-tasks. This program is also described in the chapter Tool-Programs.

The tool 'sched' calls the new function 'SetScheduling' which requires a parameter between 1 and 20. The higher the value the more low-activity tasks are preferred and the longer they get more CPU. This value is set to 6 by default.

Please note that the old priorities have no meaning anymore. But nevertheless the priority should always be set to a reasonable value.

The WarpOS-scheduler is also capable of putting the PowerPC-processor into power-save mode if no tasks want to run (or all of them are in wait-state). For the technically interested: The 'Nap'-Mode that turns off almost all units of the PPC except the CPUclock and the TimeBase is used to achieve this.

At this point you should note some important differences between exec and WarpOS: While exec allowed the programmer to switch off all interrupts (and multitasking with them), this is NOT possible with WarpOS nor is this ever used internally. Switching off multitasking without switching off the interrupts (exec/Forbid, exec/Permit) is also not possible for applications anymore and is used internally only sparingly and for very short amounts of time. Accesses to resources that are to be protected from multiple accesses must be protected by semaphores instead.

1.7 The Task-structure

When it comes to the task-structure, WarpOS also strongly adheres to the standards set by exec. Therefore the first element of the PPCTask-structure is an exec-task structure. Further WarpOS-specific elements follow, but most of them are only for internal use and not of interest to applications. Important: the size of the task-structure is subject to future changes, any assumptions regarding the size of a task-structure are illegal.

The PPCTask-structure is described in the 'tasksPPC.i' include-file. No write access should ever be made to any such structure. For version V11 of the powerpc.library, the structure is as follows:

```

STRUCTURE      TASKPPC,TC_SIZE
ULONG          TASKPPC_STACKSIZE
APTR           TASKPPC_STACKMEM           ;private
APTR           TASKPPC_CONTEXTMEM         ;private
APTR           TASKPPC_TASKPTR            ;private
ULONG          TASKPPC_FLAGS
STRUCT         TASKPPC_LINK,TASKLINK_SIZE ;private
APTR           TASKPPC_BATSTORAGE         ;private
ULONG          TASKPPC_CORE               ;private
STRUCT         TASKPPC_TABLELINK,MLN_SIZE ;private
APTR           TASKPPC_TABLE              ;private
ULONG          TASKPPC_DEBUGDATA

; These elements are new for version 14
UWORD          TASKPPC_PAD                ;private
ULONG          TASKPPC_TIMESTAMP          ;private
ULONG          TASKPPC_TIMESTAMP2         ;private
ULONG          TASKPPC_ELAPSED            ;private
ULONG          TASKPPC_ELAPSED2           ;private
ULONG          TASKPPC_TOTALELAPSED       ;private
ULONG          TASKPPC_QUANTUM            ;private
ULONG          TASKPPC_PRIORITY           ;private
ULONG          TASKPPC_PRIOFFSET          ;private
APTR           TASKPPC_POWERPCBASE       ;private
ULONG          TASKPPC_DESIRE            ;private
ULONG          TASKPPC_CPUUSAGE           ;private
ULONG          TASKPPC_BUSY               ;private
ULONG          TASKPPC_ACTIVITY           ;private
ULONG          TASKPPC_ID                 ;private
ULONG          TASKPPC_NICE               ;private
LABEL          TASKPPC_SIZE

```

All Fields marked as 'private' are to be considered as such and will not be discussed further. The additional fields of the PPCTask-structure follow below:

- TASKPPC_STACKSIZE : stack size for this task
- TASKPPC_FLAGS : some additional task flags, the following are currently supported:
 - TASKPPCF_SYSTEM : denotes a system-task
 - TASKPPCF_BAT : Set if the task runs on a pure BAT-MMU-Setup instead of a normal page table.
- TASKPPC_DEBUGDATA : This field can be used by debuggers to store their private task-specific data.
- TASKPPC_CPUUSAGE : The CPU load of this task, which defines how much percent of its time a task uses the CPU. To get the real value in percent the value has to be divided by 100.
- TASKPPC_BUSY : The time a task is not in waiting state. The value also has to be divided by 100 to get the real value in percent.
- TASKPPC_ACTIVITY : Gives the activity of a task. It's calculated by the formula

Activity = RUNNING time / (RUNNING time + WAITING time)

This value also has to be divided by 100 to get the real value in percent.

- TASKPPC_ID : The ID number of this task
- TASKPPC_NICE : The NICE value of this task

Reading task-data is only allowed between calls to 'LockTaskList' and 'UnlockTaskList'!

The PPCTask-structure begins with an exec-task structure. However, not all fields are supported. Below, you find the exec-task structure and an explanation of what fields are significant:

STRUCTURE	TC_Struct, LN_SIZE
UBYTE	TC_FLAGS
UBYTE	TC_STATE
BYTE	TC_IDNESTCNT
BYTE	TC_TDNESTCNT
ULONG	TC_SIGALLOC
ULONG	TC_SIGWAIT
ULONG	TC_SIGRECV
ULONG	TC_SIGEXCEPT
APTR	tc_ETask
APTR	TC_EXCEPTDATA
APTR	TC_EXCEPTCODE
APTR	TC_TRAPDATA
APTR	TC_TRAPCODE
APTR	TC_SPREG
APTR	TC_SPLOWER
APTR	TC_SPUPPER
FPTR	TC_SWITCH
FPTR	TC_LAUNCH
STRUCT	TC_MEMENTRY, LH_SIZE
APTR	TC_Userdata
LABEL	TC_SIZE

The first element, the node-structure for linking, has the same function as in the exec-task structure. The type of a PPC-task is new: NT_PPCTASK. Furthermore, the following fields are of significance:

- TC_FLAGS : The flags TF_SWITCH and TF_LAUNCH are supported. They have the same meaning like in exec. These callback functions can be used from version V13 of the powerpc.library.
IMPORTANT: These flags must absolutely be set AFTER the function pointers are installed (into the fields TC_SWITCH resp. TC_LAUNCH)!
- TC_STATE : Has the same meaning as in the exec-task structure. One status has been added: TS_CHANGING. This status means that a task in wait-state will change to Ready- or Running-state very shortly. This should basically be interpreted in the same way as TS_WAIT.
- TC_SIGALLOC : Has the same meaning as in the exec-task structure. Only bits 12 through 15 of the system bits are supported. Bit 10 has a

special meaning in WarpOS: it is set if a task was told to wait for a certain amount of time (function WaitTime) and this time has expired.

TC_SIGWAIT : Has the same meaning as in the exec-task structure.

TC_SIGRECVD : Has the same meaning as in the exec-task structure.

TC_SPLLOWER : Lower End of the task-stack.

TC_SPUPPER : Upper End of the task-stack.

TC_SWITCH : Has the same meaning as in the exec-task structure. If the PPC task looses the CPU, this function is called, if the appropriate flag is set. This function is called inside the scheduler (and therefore is as a part of an exception handler in supervisor mode) and gets the smalldata base of the task, which is loosing CpU, in r2. No further information is passed. The callback function has to follow the same rules as for conventional exception handlers. These callback functions can be used from V13 of the powerpc.library!

TC_LAUNCH : Same meaning like in exec. See also TC_SWITCH. Can also be used only from V13 on.

TC_MEMENTRY : A list of all allocations that are to be freed when the task is terminated. Has the same function as in the exec-task structure. ←

TC_Userdata : Has the same meaning as in the exec-task structure.

The remaining fields are currently not supported but this can change in the future.

1.8 The Task-functions

This section describes the library-functions that allow the handling of PPC-tasks. Detailed descriptions of every function can be found in the 'powerpc.doc', the information given here is of more general nature.

The following functions for task-handling are available:

- CreateTaskPPC
- DeleteTaskPPC
- FindTaskPPC
- SetTaskPriPPC
- LockTaskList
- UnLockTaskList

New PPC-task can be created using 'CreateTaskPPC'. This function is passed a taglist that contains all desired parameters. The following tags are supported (defined in the 'tasksPPC.i' include-file):

- TASKATTR_CODE : Start-address of the new task. The execution of the new tasks starts in this place.
- TASKATTR_EXITCODE : Address of a function that is executed after the task leaves the actual program by means of a conventional return. This function should then usually terminate the task. If this tag is not passed, the task jumps to the standard-exit routine of WarpOS and is terminated.

```

- TASKATTR_NAME      : Name of the task
- TASKATTR_PRI       : Priority of the tasks (-128 to 127)
- TASKATTR_STACKSIZE : Size of the PPC-stack
- TASKATTR_R2
  TASKATTR_R3
  ...
  TASKATTR_R10      : Starting values for the registers r2-r10
- TASKATTR_MOTHERPRI : A boolean-variable. If it is set, the new task
                      inherits the priority of the task it was created
                      by. TASKATTR_PRI is then ignored.
                      Since V14, the NICE values are also inherited.
- TASKATTR_BAT       : A boolean-variable. If it is set, the task runs
                      using the BAT-MMU-setup instead of the conventional
                      page-table. Should usually not be set.
- TASKATTR_NICE      : The NICE value of a new task (new since V14)

```

If the function is successful, a pointer to the created task is returned, otherwise NULL.

A task can be removed using 'DeleteTaskPPC'. Theoretically, any task can be terminated, but it should be avoided to let tasks terminate anything but themselves. A task is also terminated if the main program is left by means of a conventional return-command.

Since V14, all the memory allocated by a task during its activity is freed at task removal

Finding a task is achieved by calling 'FindTaskPPC' which has the same function as the corresponding exec-function 'FindTask'. This function is most often called with a null-parameter to determine the current task, which, as with exec/FindTask, happens very quickly.

Since V14 there is another function to find a task, but this function requires a task ID number as parameter. This function is called 'FindTaskByID'.

'SetTaskPriPPC' is used to assign a PPC-task a new priority. Depending on the value the task might even start running immediately.

To obtain information about PPC-tasks the 'LockTaskList' and 'UnLockTaskList' functions can be used. 'LockTaskList' allows exclusive access to a list of all tasks without halting the multitasking. However, no new tasks are generated until the access is freed using 'UnLockTaskList'.

'LockTaskList' returns a pointer to a TaskPtr-structure which looks as follows:

```

STRUCTURE      TASKPTR, LN_SIZE
APTR           TASKPTR_TASK
LABEL         TASKPTR_SIZE

```

The head of the structure is formed by a node that links several structures. TASKPTR_TASK is a pointer to a PPCTask which can now be read in this way. In order to get to the next task, you jump to the next TaskPtr-Structure until you reach the end of the list. After evaluation the access to this list

should be freed again using 'UnLockTaskList'.

1.9 Context Switches

One of the main purposes of WarpOS is to ensure quick and efficient communication between the two processors. It provides functions to call a function on the respective "other" CPU.

An application that uses the PowerPC always consists of a 68K-part and a PPC-part. The CPUs are switched every time a function which makes such a switch necessary is called.

For this reason an application runs on both CPUs, but sequentially (as long as there aren't several tasks involved). If it runs on the 68K an appropriate 68K-task is running, if it runs on the PPC an appropriate PPC-task is running. Any such program will always consist of two task that are closely tied to each other. These tasks are usually referred to as "mirror tasks".

While one task is running, the other one is waiting for the current task to perform a context switch. In this case the running task makes contact to his partner task (by sending a signal) and passes all necessary parameters after which the partner executes the next function. This switch is done extremely fast because the communication occurs directly between the mirror tasks and is not handled by a server task.

The waiting task has a few other duties to fulfill, too. Among other things, it passes incoming signals to its mirror task. This way, it is e.g. possible for a PPC-Task to wait for the CTRL-C/D/E/F signals.

When a new mirror task is created (e.g. when a PPC-task is created) the mirror tasks are assigned the same name plus an extension. For 68k-mirror tasks this extension is '_68K', for PPC-mirror tasks it is '_PPC'. If a mirror-task of a shell-process is created, the process number is also attached to the mirror task. Example: Shell Process_PPC7.

Since V14, the name building of mirror tasks has changed for CLI background tasks. Now the name of the executed program is used instead of the process' name. But the CLI number is still put at the end of the PPC tasks name.

If a context switch occurs, the appropriate functions of the powerpc.library take care that the necessary provision are taken, which means first and foremost that all caches are flushed. In this way the notorious cache-conflicts between the two processors can be avoided. This also means that a task and its mirror task can access the same global variables. As soon as several tasks come into play you should be really careful as this requires a lot of experience with dual-processor-systems.

The powerpc.library provides two functions for context switches: RunPPC and Run68K. RunPPC is a function for the 68K-processor and jumps to a PPC-function. All registers as well as areas on the stack can be passed in this case, although passing stack areas is relatively complex due to the complicated structure of the PPC-stack. However, most of the time the register parameters will suffice. Additionally, RunPPC offers the possibility to execute PPC-functions asynchronously but this will not be covered further as it should usually not be done because it is quite difficult and results in a

number of restrictions.

Run68K works exactly the same way for the opposite case that a 68K-function is to be called from the PPC. This is very often necessary to call AMIGA-OS functions which are not available in native code. Run68K demands the same structure as RunPPC and is used virtually identical.

If programs are developed using development environment such as 'StormC' the programmer need not concern himself with context switches at all. He can call a function that is intended for the "other" CPU like any other function - the compiler and linker then automatically take care that the context switch via RunPPC or Run68K is made.

The structure passed to RunPPC or Run68K looks as follows:

STRUCTURE	PP,0
APTR	PP_CODE
ULONG	PP_OFFSET
ULONG	PP_FLAGS
APTR	PP_STACKPTR
ULONG	PP_STACKSIZE
STRUCT	PP_REGS,15*4
STRUCT	PP_FREGS,8*8
LABEL	PP_SIZE

Regarding the fields:

PP_CODE	: A pointer to the function to be executed. If a library-function is to be executed from the PPC, the library-base of the respective library is placed here. It's also possible to call PPC library functions directly from the 68K. The address of the code of the corresponding function has to be calculated by the following formula and written into the element PP_CODE: Code-address = library base + library offset + 2
PP_OFFSET	: An offset that is added to PP_CODE. Only supported by Run68K until version 12.2 of powerpc.library. When making library-calls the library-offset of the respective function is placed here. From version 12.3 on, this field is also supported by RunPPC in the same manner as by Run68K.
PP_FLAGS	: possible flags: PPF_ASYNC : function is executed asynchronously
PP_STACKPTR	: Pointer to the upper end of the stack area to be passed (0 if no stack area is to be passed)
PP_STACKSIZE	: Size of the stack area to be passed (0 if no stack area is to be passed)
PP_REGS	: An array of 15 longwords to pass the registers d0-a6 The assignment of the registers can be found in 'powerpc.doc'
PP_FREGS	: An array of 8*8 Bytes to pass double-precision floating-point registers.

1.10 Signaling

Signaling in WarpOS works analogous to exec. Just as in exec there are 16 task-signals, the upper 16 of which are accessible to applications. The CTRL-C/D/E/F system signals are supported. On top of that, bit 10 has an additional function when used together with the 'WaitTime' function. Unlike exec, WarpOS does not support SoftInt-signals but this will possibly change in the future.

The 'AllocSignalPPC', 'FreeSignalPPC', 'SignalPPC', 'SetSignalPPC' and 'WaitPPC' functions are exact equivalents of the respective exec-functions. Additionally, there is another function to explicitly send a signal to a 68K-task: 'Signal68K'. This function is used in the exact same way as 'SignalPPC'.

Furthermore, there is a function that allows waiting for signals (like 'WaitPPC' does) but additionally supports a timeout-value. As soon as the preset time has passed, the 'WaitTime' function is left. If this happened because the preset time has expired, the signal mask returned is set to 0.

An important feature of WarpOS is the support for virtual signals. All signals are shared between a task and its mirror task. This is best understood if you view the 16 signal bits as not belonging to a certain task but rather to a certain application that consists of two tasks (one 68K-task and one PPC-task). Depending on which CPU the task is running on, the signal sets of the 68K-task or the PPC-task are used. These distinct signal sets are always equivalent to the 16 virtual signal bits, however. If a task allocates a signal it is also allocated for the mirror task.

This approach has a number of advantages. On the one hand, all signals that arrive at one task are passed on to its mirror task. If the mirror task is currently running the signals are sent, otherwise they are passed on when the next context switch occurs. For the programmer this has the effect that he can simply send a signal to an application and it will automatically be passed to the correct CPU.

This goes even further: It is even possible to send signals directly to a PPC-task by ways of the standard exec-signaling functions - you simply have to pass a pointer to the PPC-task instead of a pointer to a 68K-task to the 'Signal' function. This also works in reverse order: the WarpOS-function 'SignalPPC' can be used to send signals directly to 68K-tasks. This in turn also means that each CPU can send signals through its native functions or can wait for signals, regardless where the signals are going or are coming from.

The essence of all this is: it does not matter what CPU your application runs on. The signals are always passed to the correct recipient. Based on this CPU-independent signaling, very fast communication can be achieved.

1.11 Message-Handling

The message-handling also works virtually identical to that of exec. A first difference to exec can be found in the MessagePort-structure for the PPC. The first element still is a standard-exec-MessagePort. A number of other fields not of interest to the programmer have been added, however. These MessagePorts should never be accessed directly - instead, the provided library-functions should be used to do so.

The following is a list of message-handling functions provided by WarpOS:

```
CreateMsgPortPPC
DeleteMsgPortPPC
AddPortPPC
RemPortPPC
FindPortPPC
WaitPortPPC
PutMsgPPC
GetMsgPPC
ReplyMsgPPC
SetReplyPortPPC
```

CreateMsgPortPPC is equivalent to exec/CreateMsgPort. It creates a PowerPC-MessagePort. Unlike in exec, this is the only permitted way to create a PowerPC-MessagePort.

The counterpart is DeleteMsgPortPPC which frees a previously generated PowerPC-MessagePort.

AddPortPPC and RemPortPPC are used to make MessagePorts publicly accessible, again in direct correspondence to exec. They are managed in a separate list, i.e. they are not found in the same list as the public 68K-MessagePorts.

FindPortPPC can be used to find any public port the name of which is known (same function as in exec). Unlike in exec, calls to FindPortPPC need not be protected by semaphores – this is done automatically.

WaitPortPPC, PutMsgPPC, GetMsgPPC and ReplyMsgPPC are the functions known from exec to send, receive, and respond to messages. Their use is identical to that of the respective exec-functions. The structure of the messages has remained unchanged.

Important: It is not allowed to access ports that belong to a different CPU.

However, since version V12 it is also possible to send messages to non-native message-ports. It may become necessary to use new functions to do this.

To send a message to the other CPU, it must first be allocated using the 'AllocXMsg' (68K) or 'AllocXMsgPPC' (PPC) function. These functions take care that the message is properly initialized and has the required alignment.

After allocating such an inter-CPU-message, the actual message content can be transmitted in the allocated message.

Now the message can be transmitted to the other CPU using the 'PutXMsg' (68K) or 'PutXMsgPPC' (PPC) function. For this purpose, the function must be passed the address of a message port that belongs to the other CPU.

After the message has been responded to by the other CPU, it can be either used again or freed using the 'FreeXMsg' (68K) or 'FreeXMsgPPC' (PPC) function.

Receiving, reading, and responding to messages is handled by the standard

message-handling mechanisms of `exec.library` (68K) or `powerpc.library` (PPC). You should note that the inter-CPU-messages are given a different node-type than normal messages. If you want to distinguish sent messages from messages that have been responded to, you must compare the node-type to `NT_REPLYMSG`. Inter-CPU-messages that have been responded to further receive the node-type `NT_REPLYMSG`. Any assumptions regarding the value of the new node-type are illegal.

The recipient of an inter-CPU-message may not access any data not directly contained within the message (e.g. any data referred to by pointers) unless both tasks have taken the necessary precautions regarding cache-consistency. For the actual message-data contained in the message this is done automatically.

The recipient of an inter-CPU-message may write into a message and respond to it after that.

The `'powerpc.doc'` document contains further notes covering the use of functions for `'inter-CPU-message-handling'`.

In the version V13 of the `powerpc.library` a new function was added: `'SetReplyPortPPC'`. This function allows to exchange the reply port of a message. It gets the message and the new reply port as input parameter and returns the old reply port as output parameter. This function can be used for both conventional PPC messages and for Inter-CPU messages.

1.12 Memory Management

The memory management of WarpOS works hand in hand with that of `exec`. In principle, WarpOS allocates large memory areas from `exec` which are then managed locally. Therefore, allocations from the PPC-side are handled completely native most of the time.

The WarpOS memory management offers some powerful additional features. User-defined alignment is supported as well as many features of the PPC-MMU. On top of that you can always be sure that memory areas have a minimum alignment of 32 in order to avoid cache-conflicts.

Version V12 and up support facultative memory protection. Tasks have the possibility to allocate memory which is protected from access by other tasks.

I would like to take this opportunity to once again stress the fact that it is illegal to do PowerPC-write accesses to memory areas that have been allocated from the 68K-side if the alignment is not at least 32 (for the beginning AND the end of the area). Otherwise there is acute danger that the PowerPC overwrites areas before and after the desired memory area.

In order to avoid this problem, version V12 and up offer the `'AllocVec32'` and `'FreeVec32'` functions for the 68K-processor. They work exactly like the `'AllocVec'` and `'FreeVec'` functions of `exec.library` but take care that the memory area has the necessary alignment to be able to share it with the PPC-task.

Any time memory is allocated for the PPC using the WarpOS memory management, at least 64 bytes of memory are needed regardless of the actual amount of

memory allocated. For this reason it is not advisable anymore to allocate lots of memory in small chunks.

Currently, the following memory-management functions are available:

```
AllocVecPPC
FreeVecPPC
FreeAllMem
```

AllocVecPPC is equivalent to exec/AllocVec but offers some additional features. It supports an additional parameter that describes the desired alignment. If necessary, this value will also be rounded up internally.

AllocVecPPC knows additional memory attributes which allow the allocation of memory areas using certain cache modes and utilize the features of the MMU. These additional modes should not be used carelessly but only if you want to create highly optimized software. First and foremost this applies to games and demos which are able to take huge advantage of this. The best example is the 'voxelspace' Demo-Program which gains a lot from these features. Further tips for optimally exploiting this feature can be found in the 'GameDev.guide' document.

The additional memory attributes have the following meanings:

- MEMF_WRITETHROUGH : Memory is allocated using the 'cachable writethrough' cache mode. In this mode all write accesses are done directly in memory.
- MEMF_COPYBACK : Memory is allocated using the 'cachable copyback' cache mode. In this mode all write accesses are done on the cache. This is usually the standard setting.
- MEMF_CACHEON : The cache is switched on for this area of memory. This is usually the standard setting.
- MEMF_CACHEOFF : The cache is switched off for this area of memory.
- MEMF_GUARDED : The memory area is given the 'guarded' attribute.
- MEMF_NOTGUARDED : The memory area is given the 'not guarded' attribute.
- MEMF_BAT : The memory area is controlled by a BAT-register. BAT-registers define address translations for very large areas of memory (comparable to the transparent translation registers of the 68K). The advantage of BAT-registers is that no table searches need to be performed - this could cause a large performance problem otherwise. The disadvantage: this needs a lot more memory than effectively desired. On top of that, the number of BAT-Register for each task is limited to 4.
- MEMF_PROTECT : The allocated memory area is protected from access by other tasks.
- MEMF_WRITEPROTECT : The allocated memory area is protected from write access by other tasks.

Single memory areas can be freed using 'FreeVecPPC'. The function is effectively identical to exec/FreeVec.

Another useful function is 'FreeAllMem' which frees all memory previously

allocated by the task it is called from.

1.13 Semaphores

Semaphores are a lot more important in WarpOS than they used to be in AMIGA-OS. In AMIGA-OS, programmers took the easy way out all too often and enabled exclusive access to resources by switching off multitasking or even the interrupts. Both isn't possible in WarpOS anymore and thus this task has to be taken over by semaphores.

One difference between WarpOS and exec is the structure of semaphores which was extended in WarpOS by an additional field which is only used for internal purposes.

WarpOS offers the functions know from exec which are 100% identical in function:

```
InitSemaphorePPC
AddSemaphorePPC
RemSemaphorePPC
FindSemaphorePPC
ObtainSemaphorePPC
AttemptSemaphorePPC
ReleaseSemaphorePPC
```

A significant difference between WarpOS and exec: A semaphore created using InitSemaphorePPC should be freed using 'FreeSemaphorePPC' because InitSemaphorePPC (unlike exec/InitSemaphore) allocates memory which should be freed again.

1.14 Lists / Tag Lists

WarpOS offers equivalents to all functions for list management known from exec. Most of these are also contained as assembler-macros in the 'listsPPC.i' include-file. The parameters are the same for both versions. When using the assembler-macros special attention should be paid to which registers are overwritten.

The following library-functions for list management are supported:

```
InsertPPC
AddHeadPPC
AddTailPPC
RemovePPC
RemHeadPPC
RemTailPPC
EnqueuePPC
FindNamePPC
```

These library-functions are different from the remaining ones in the respect that they are guaranteed to function even if the library base is not passed in r3. As a general rule, the base must be passed along with any library function

and it should always be assumed to be necessary so unless stated differently (as is the case here).

A note on FindNamePPC: This function is NOT protected by semaphores!

Along with the list funtions there are also function to handle taglists as they are know from utility.library:

```
FindTagItemPPC
GetTagDataPPC
NextTagItemPPC
```

1.15 Hardware Interfaces

WarpOS offers a number of functions that can be used to execute functions that are very close to the hardware. The topic Exception Handling will be covered later in a separate chapter.

WarpOS offers the option of switching into supervisor-mode. In supervisor-mode an application has access to the entire hardware. Applications should usually not switch into supervisor-mode. Instead, library functions that fulfill the same role should be used whenever possible.

The 'Super' function switches into supervisor-mode while 'User' switches back into user-mode. 'User' must not be called in user-mode because otherwise a privilege-violation is caused.

A function for cache management is of course also necessary. The 'SetCache' function allows pin-point manipulation of the caches. It expects a mode-value as parameters which more thoroughly describes the task to be performed. Some tasks can be performed on the entire cache as well as on a limited address range. Others can only be performed either on the entire cache or on an address range. An address area is passed by specifying its starting address and length while a starting address and length of zero signify the entire cache.

The following modes are supported (found in the 'powerpc.i' include-file):

CACHE_DCACHEOFF	: The data-cache is switched off.
CACHE_DCACHEON	: The data-cache is switched on.
CACHE_DCACHELOCK	: The data-cache is frozen. The content of the data-cache remains the same until it is put back into normal state. This mode only works if an address range was specified. In this case the specified area is copied into the cache and the cache is frozen after that.
CACHE_DCACHEUNLOCK	: Reverts the data cache to normal state after freezing.
CACHE_DCACHEFLUSH	: All data that is not yet located in the main memory is written into memory. This mode can be used on the entire cache as well as on an address range.
CACHE_ICACHEOFF	: The instruction-cache is switched off.
CACHE_ICACHEON	: The instruction-cache is switched on.
CACHE_ICACHELOCK	: The instruction-cache is frozen. This mode can only be used on the entire cache.

CACHE_ICACHEUNLOCK : Reverts the instruction cache to normal state after freezing.

CACHE_ICACHEINV : The instruction-cache is invalidated. This mode can be used on the entire cache as well as on an address range.

The 'SetHardware' function allows the activation and de-activation of modes that are close to the hardware. Similar to 'SetCache' it expects a mode value which more thoroughly describes the task to be performed. Depending on the mode a further parameter may be required. This function returns a status value which states whether the present CPU supports the function.

The following modes can be specified:

HW_TRACEON : The trace-mode is switched on. In trace-mode a trace-exception is triggered every time a command has been executed. You should note that this mode is already activated before 'SetHardware' terminates. To turn on trace-mode at an exact time, it must be manually switched on in an exception-handler.

HW_TRACEOFF : The trace-mode is switched off.

HW_BRANCHTRACEON : The branchtrace-mode is switched on. After every the execution of any jump command an exception is triggered.

HW_BRANCHTRACEOFF : The branchtrace-mode is switched off.

HW_FPEXCON : The floating-point-exceptions are switched on. This is only the global On-/Off-switch. Additionally, the 'ModifyFPExc' function must be called to toggle the single FEp-Exceptions.

HW_FPEXCOFF : The floating-point-exceptions are switched off.

HW_SETIBREAK : The instruction-breakpoint is set. An additional parameter has to be passed which specifies the address of the breakpoint to be set. As soon as the PPC executes the command located at this address, an instruction breakpoint exception is triggered.

HW_CLEARIBREAK : The instruction-breakpoint is switched off again.

HW_SETDBREAK : The data-breakpoint is set. Also see HW_SETIBREAK. As soon as an access is made to the specified address, a data access exception is triggered. The data-breakpoint is not supported by all PPC-CPU's.

HW_CLEARDBREAK : The data-breakpoint is switched off again.

If the floating-point-exceptions are switched on using the 'SetHardware' function, you have to further specify which exceptions you want to occur. There are five different kinds of FP-exceptions. Using the 'ModifyFPExc' function you can toggle single FP-exceptions at your leisure.

This functions demands a bitmask and can thus toggle several exceptions at the same time. The following flags are supported:

FPE_EN_OVERFLOW : Switches on the overflow-exception

FPE_EN_UNDERFLOW : Switches on the underflow-exception

FPE_EN_ZERODIVIDE : Switches on the exception for division by zero

FPE_EN_INEXACT : Switches on the exception for unprecise operations

```

FPF_EN_INVALID      : Switches on the exception for invalid operations
FPF_DIS_OVERFLOW    : Switches off the overflow-exception
FPF_DIS_UNDERFLOW   : Switches off the underflow-exception
FPF_DIS_ZERODIVIDE  : Switches off the exception for division by zero
FPF_DIS_INEXACT     : Switches off the exception for unprecise operations
FPF_DIS_INVALID     : Switches off the exception for invalid operations

```

A special function is 'ChangeMMU'. This function is different from the others in that way that it is not of global character but only affects the calling task. This function has two modes (defined in the 'tasksPPC.i' include-file):

```

CHMMU_STANDARD      : The task runs using the conventional MMU-setup using
                      a pagetable. This is usually the default.
CHMMU_BAT            : The task runs using a special BAT-based MMU-Setup.

```

This function should not be used unless a lot of know-how regarding MMUs is present and even then the step should be well considered.

The 'GetInfo' function returns a lot of information about the hardware present. It demands a taglist that contains a list of the desired information. After execution, the desired information can be found in the corresponding 'ti_data' field of the taglist.

The following tags are supported (to be found in the 'powerpc.i' include-file):

```

PPCINFO_CPU          : The CPU-type is stated as a bitmask which only has
                      one bit set. The following flags are supported:
                      CPUF_603      : PowerPC 603
                      CPUF_603E     : PowerPC 603E
                      CPUF_604      : PowerPC 604
                      CPUF_604E     : PowerPC 604E
                      CPUF_620      : PowerPC 620
PPCINFO_PVR          : The value of the PVR-register is returned.
PPCINFO_ICACHE       : The status of the instruction-cache is returned
                      (as a bitmask). The following flags are supported
                      CACHEF_ON_UNLOCKED : Cache is switched on and not
                                          frozen
                      CACHEF_ON_LOCKED   : Cache is switched on and
                                          frozen
                      CACHEF_OFF_UNLOCKED : Cache is switched off and
                                          not frozen
                      CACHEF_OFF_LOCKED  : Cache is switched off and
                                          frozen
PPCINFO_DCACHE       : The status of the data-cache is returned. The
                      possible flags are the same as for PPCINFO_ICACHE

PPCINFO_PAGETABLE    : The address of the standard-system-pagetable is
                      returned. This pagetable is used by all tasks that
                      have not allocated any protected memory.

PPCINFO_TABLESIZE    : The size of the current pagetable is returned.
PPCINFO_BUSCLOCK     : The bus clock is returned.
PPCINFO_CPUCLOCK     : The processor clock is returned. This function is
                      not available for processors that do not know the

```

HID1-register (Zero is returned in this case). The PPC603 and PPC604 processors are affected by this. In case of the AMIGA the PPC603E and PPC604E are used.

Since V14 there exists a function which allows to retrieve status information from the underlying HAL (the WarpUp-HAL). The function 'GetHALInfo' requires a taglist, that contains a list of the desired information. After execution, the desired information can be found in the corresponding 'ti_data' field of the taglist.

The following tags are supported (to be found in the 'powerpc.i' include-file):

HINFO_ALEXC_HIGH	: The upper longword of a 64 bit value is returned, which states how many emulated alignment exceptions occurred since the PowerPC was reset. These exceptions always occur if floating point values are accessed on addresses which are not divisible by 4. The WarpUp-HAL emulates those exceptions and therefore cares that the application doesn't crash.
HINFO_ALEXC_LOW	: The lower longword of the above mentioned 64 bit value is returned.

1.16 Exception Handling

With WarpOS, applications are able to install their own exception handlers in the system. This is done using the 'SetExcHandler' function which is in a way comparable to the 'AddIntServer' exec-function. The WarpOS-function offers a number of additional features. For further information see 'powerpc.doc'.

Exception handlers can be global or task-specific. Task-specific exception handlers are only executed if a certain task caused the exception.

Exception handlers may cover several exception types or just one.

WarpOS supports priorities for exception handlers. The higher the priority of an exception handler is, the sooner it is executed. An exception handler can also use the return value to determine if exception handlers of lower priority should be executed at all. If not, the exception is left immediately. This can be done to e.g. perform emulations.

If no exception handler breaks off the exception and passes the ball back to the interrupted program, the standard-exception handler of WarpOS is invoked which opens a big requester and prints a lot of information on the crash.

There are two kinds of exception handlers: LowLevel and HighLevel-exception handlers. The programmer can choose either of the two types when calling 'SetExcHandler' and then has to implement the actual function accordingly.

LowLevel-exception handlers are usually passed the unchanged register contents of the interrupted program. Exceptions to this rule are:

r3 is passed in the XCONTEXT structure which is then passed to the exception handler through r3 (see 'powerpc.i' include-file). The link-register of the interrupted program is passed in SPRG1. The stack pointer (r1) of the

interrupted program is passed in SPRG2. The smalldata-base (r2) of the interrupted program is passed in SPRG3.

LowLevel-exception handlers are suited for time-critical tasks such as emulations and can usually only be written in assembler. Care has to be taken that no registers are overwritten which must not be overwritten (this includes SPRs).

HighLevel-exception handlers are passed all values of the interrupted program in the EXCCONTEXT-structure. If values are to be changed, the appropriate values within the structure must be changed. The handler can be an ordinary function and can also be written in a high-level programming language.

The 'SetExcHandler' function is passed a taglist. The following tags are supported for this:

EXCATTR_CODE	:	The address of the exception handler
EXCATTR_DATA	:	A value that is passed to the exception handler in r2 (usually a base that allows access to the global variables).
EXCATTR_TASK	:	Defines for which interrupted task the exception handler should be executed. Zero signifies the current task. This attribute is ignored if the exception handler is a global one.
EXCATTR_EXCID	:	A bitmask that defines which exceptions this exception handler should execute. The 'powerpc.i' include file lists all possible flags.
EXCATTR_FLAGS	:	Possible Flags:
		EXCF_GLOBAL : The exception handler is global, this means it is task independent.
		EXCF_LOCAL : The exception handler is task-specific. (also see EXCATTR_TASK).
		EXCF_SMALLCONTEXT : The exception handler is a LowLevel-handler.
		EXCF_LARGECONTEXT : The exception handler is a HighLevel-handler.
EXCATTR_NAME	:	Name of the exception handler
EXCATTR_PRI	:	Priority of the exception handler (-128 to 127).

The 'powerpc.doc' document contains a number of notes on 'SetExcHandler' that should be read and understood before using the function.

An exception handler created using 'SetExcHandler' can be removed by using 'RemExcHandler'. It requires the return value of 'SetExcHandler' as a parameter.

Exception handlers are usually called with the MMU switched off. The pagetable must not be switched on as this can lead to tremendous problems if a CPU without hardware-tablesearch is present in the system. If the MMU is switched off, all memory access is handled in copyback-mode. This leads to problems when accessing critical memory areas as the CustomChip-area. Access to these areas should always be made in 'noncachable' mode.

In order to remedy this problem there are two functions that may only be called from within exception handlers: 'SetExcMMU' and 'ClearExcMMU'.

The first function install a temporary MMU-setup in the BAT-registers and takes care that every memory area is accessed using a reasonable cache-mode.

'ClearExcMMU' reverts this procedure.

1.17 WarpOS-Debugger

WarpOS contains an integrated debugging-system that can support the author of WarpOS as well as developers of PPC-applications a great deal. It is implemented as a logging-system that writes a lot of information on the serial port. For example, entering and leaving most of the system functions is logged. If only the entering is logged this leads to the conclusion that the program crashed within this system function. In this way, pin-pointing the location of errors is made a lot easier.

In order to really work efficiently you should install a program that intercepts output to the serial port and displays it in a window. The most common program of this kind is most likely to be 'sushi'.

The logging-system has three steps. Depending on which step is activated, more or less information is sent to the serial port. This debugging-level can be set by using the 68K-library-function 'PowerDebugMode'. It requires a number between 0 and 3 as parameter. A value of 0 completely switches off the debugging-output.

There also is a tool-program called 'setdb' which can be invoked from the shell. This program also requires a number between 0 and 3.

PPC-applications can also write their own log-files on the serial port. For this purpose WarpOS provides the 'Sprintf' function. It accepts and requires the same parameters as the well-known 'printf' C-function and outputs the text to the serial port..

From version V12 on there also is an analogous function for the 68K-processor. 'Sprintf68K' which does exactly the same as 'Sprintf' for the PPC.

From version V13 on, the possibility exists to monitor the beginning of new PPC tasks and the removal of themselves. Using the function 'Snooptask', a callback job can be installed, that means, that the callback function specified is called every time, when a new PPC task is created resp. a PPC task is removed. Further information is found at the description of this function in the autodocs.

The function 'EndSnoopTask' removes a callback job, which was installed using 'SnoopTask'.

1.18 Preferences

WarpOS knows some env-variables that can be used to make certain preferences. The following are currently supported:

env:powerpc/debug

May contain a number between 0 and 3. This specifies the debugging-level to use when WarpOS boots. Should usually be left at 0 unless you are interested

in what happens during the booting process. The debugging-level can be changed during operation by using the 'setdb' tool.

env:powerpc/crashfile

Demands a filename/path. If a PPC-crash occurs, WarpOS writes all crash data to this file. The file may also be a CON-Window in order to display the data in a window.

If the file string is preceded by a semicolon no output is made. This is useful for people running programs such as 'sushi' which also display the crash data. This way you can keep the data from appearing twice.

env:powerpc/alertfile

Works in the same way as the 'crashfile' variable only that this file is used for system-messages (e.g. corrupt semaphores).

env:powerpc/memprot

Can be either 0 or 1. If it is set to 1, the memory-protection facilities of WarpOS are switched on, otherwise off. If low-memory situations occur it can help to switch off memory protection.

env:powerpc/gfxaddr

Requires a hexadecimal address (either with or without prefixed \$). If the powerpc.library isn't able to find the graphics-RAM, it takes the value from this environment variable. If an expansion address space is found which contains this address, the address space is put into a BAT register, if possible, to gain a higher access speed.

The variable also can be set to 0, if no problems with the graphics RAM occur.

env:powerpc/noPPC

This variable must be set, if the powerpc.library V8+ is installed on a Non-PPC system. Afterwards all demos which also run without PPC (but which try to open the powerpc.library) should run. If the variable is not set, then the library crashes at initialization.

env:powerpc/boardtype (removed since V14)

WarpOS tries to detect automatically which board is installed. If this method fails (i.e. if a alternative 68060.library is used or if other problems with 68060.library occur) the board type can be specified manually using this variable. Possible values:

- 0 - Automatic detection
- 1 - PPC developerboard (603E/150)
- 2 - CyberstormPPC board

env:powerpc/earlyterm

If this variable is not set, then problems can occur in conjunction with programs like WShell. In this case the variable should be set. Afterwards,

the tools 'stackppc' and 'changemmu' can NOT be used anymore. The PPC stack can be set using the stack command (the PPC stack gets about double the size of the 68K stack of the shell).

env:powerpc/Terminator

If this variable is set to 1, then it's possible to launch WarpOS even if the ppc.library is active. WarpOS tries then to remove the ppc.library and related resources from the operating system. Of course it is still not possible to use both ppc.library and WarpOS at the same time.

env:powerpc/HideWarning

If this variable is set to 1, the appearance of the requesters, which come up in conjunction with ppc.library Warning, is avoided.

1.19 Tool-Programs

The WarpOS-archive contains a number of useful tool-programs that are described below. For most of the tools the source code is also supplied, most of which is written in PPC-assembler.

The following tools are currently available:

```
setdb
dcoff
dcon
ibreak
dbreak
stackppc
showtasks
showinfo
changemmu
showHALinfo
GetDriverInfo
sched
stat
killppc
niceppc
```

The 'setdb' program allows to set the debugging-level of WarpOS. The higher this debugging-level, the more information is written on the serial port. This often is very helpful when trying to locate errors in the process of debugging a program.

The template of the command is:

```
setdb  LEVEL/N
```

LEVEL : A number between 0 and 3. A value of 0 means that the debugging-mode is switched off. This is also the default.

The 'dcoff' program switches off the PPC data-cache. This command has no parameters.

The 'dcon' program switches on the PPC data-cache. This command has no parameters.

The 'ibreak' program sets or deletes the instruction-breakpoint. Every PPC knows one such global breakpoint. If it is set to a certain address, executing a command at this address triggers an instruction breakpoint exception.

The template of the command is:

```
ibreak ADDRESS
```

ADDRESS : A hexadecimal that is interpreted as the breakpoint-address. The number may be preceded by a \$-sign. If the parameter is omitted, the current instruction-breakpoint is deleted.

Examples: ibreak \$8a00404
 ibreak 8000C

The 'dbreak' program sets or deletes the data-breakpoint. Any access to the specified address then triggers a data access exception. This feature is only available on the PPC604 and PPC604E. If a different CPU is present, an appropriate error message is generated.

The template of the command is:

```
dbreak ADDRESS
```

ADDRESS : An address of the same format as for the 'ibreak' command. If no parameter is specified, the current data-breakpoint is deleted.

The purpose of the 'stackppc' command is to enlarge the PPC-stack of a shell-process. It has the following template:

```
stackppc       SIZE/N
```

SIZE : The new size of the PPC-stack. If the specified size is smaller than the current stack size, the stack size remains the same. If the parameter is omitted, the program returns the current stack size.

If a new stack size is specified, a new stack is allocated and the content of the old stack is copied there. On top of that all stackframes are adapted so that they are properly linked again. The old stack is not freed.

The 'showtasks' tool shows all data of all currently installed PPC-tasks. It has no parameters. The output for one task looks like this:

Task name : Name of the PPC-task

```

Task ID           : The ID number of this task
Task location     : Address of the PPC-task structure
Task type        : Signifies the type of a task
                  SYSTEM : The task is a system-task
                  CUSTOM  : The task is not a system-task
Task state:      : Signifies the current state of the task:
                  RUNNING : The Task that executed the 'showtasks'
                           command
                  READY   : Task is active
                  WAITING  : Task is in wait-state
Task priority:   : Priority of the task
NICE value:      : NICE value of this task
MMU setup:       : Signifies the MMU-setup for this task:
                  PAGED MMU SETUP : conventional MMU setup
                  BAT SETUP       : special BAT setup

Page table location : States the address of the pagetable that is valid
                      for the current task

Stack size:        : Size of the task-stack
Stack location     : Lower end of the task-stack
Signals allocated  : Bitmask of all allocated signals
Signals to wait    : Bitmask of all signals that are currently waited
Signals received   : Bitmask of all signals that were received

```

The 'showinfo' program returns a lot of information on the hardware present in the system. It fully exploits all possibilities of the 'GetInfo' library function. The following information is displayed:

```

CPU           : The processor-type (and in parenthesis the content
                of the PVR-register)
CPU clock     : The processor clock. If the processor doesn't know
                the HID1-register, zero is displayed instead.
Bus clock     : The bus clock
Instruction Cache : The state of the instruction-cache. Shows whether
                the cache is switched on or off and whether it is
                frozen or not.
Data Cache    : The state of the data-cache. The output is the same
                as that for the instruction cache.
Page table location : The address of the standard-system-pagetable which is
                used by all tasks that haven't allocated any
                protected memory.
Page table size  : The size of the pagetable.
Time base content : The content of the time base.
CPU load        : The total CPU load in percent.
System load     : The total system load in percent. This value states
                how many CPU's would be at least necessary to execute
                all running tasks at full speed. A sytem load of 500
                percent means that at least 5 CPU's would be
                necessary.

```

The 'changemmu' program can be used to change the MMU-mode of the current shell task. It has the following template:

```
changemmu      S=STANDARD/S,B=BAT/S
```

S=STANDARD : The task runs using a conventional MMU-setup with a pagetable. This is usually the default.

B=BAT : The task runs using a special BAT-MMU-setup. This can lead to increased (but also to decreased) performance.

If no parameters are given, the current MMU-mode is displayed.

The program 'showHALInfo' prints out all status information of the underlying HAL (the WarpUp-HAL). At the moment, only the number of emulated alignment exceptions are printed out. These exceptions always occur, if floating point values on a non-4-byte-boundary are accessed. The output is done in two parts, the upper longword of a 64 bit value and the lower longword.

The tool 'GetDriverInfo' can be started without parameter. It prints out, which WarpUp hardware driver is currently installed and which version of the hardware driver protocol is supported. It is very important, that the correct hardware driver is installed, and this tool provides a possibility to test that.

The program 'sched' requires a numeric parameter between 1 and 20. The higher the value the more low-activity-tasks are preferred against high-activity tasks and the longer they get more CPU time.

The tool 'stat' prints out statistics about all PPC tasks and also about the whole system. First the program requests several times the statistical values to get reliable results by building the average value. The measuring interval is 100 milliseconds and 10 measurements are performed by default. By specifying a numeric parameter, this value can be varied between 1 and 20.

The output shows first all tasks with their statistical values:

```
- Task name      : The name of the task
- ID            : The ID number of the task
- NICE          : The NICE value of the task
- CPUUsage      : The CPU usage of this task in percent
- Busy          : The time in percent, in which the task is not
                  in waiting state
- Activity      : The activity of this task in percent
```

At the bottom there are some global parameters:

```
- CPU load       : The total CPU load of the system in percent
- System load   : The total system load in percent. A value of 100
                  percent means, that one CPU would be fully loaded
                  if everything would work in an optimal way. The
                  system load states how many CPU's would be necessary
                  to execute all tasks at full speed.
```

The tool 'killppc' can be used to remove PPC tasks. It requires the task ID as parameter. This ID can be evaluated by the tool 'stat' (see Tool-Programs).

Removing PPC tasks can be useful if crashed tasks should be eliminated. Since V14 all the memory which was allocated by this task using the WarpOS memory allocation functions are freed.

You should never remove the mirror task of a shell while the shell is working on the 68K side!

The program 'niceppc' allows to set the NICE values for a specific task. The NICE value is used to determine the priority of a task and the CPU usage. The higher the value the lower the priority and the less CPU time is given to this task. Values between 1 and 20 are legal.

The program know two parameters. The first one ('ID') allows to specific the task ID of the task to be affected. The second parameter is the NICE value. If no task is given, the current task is affected. Note that the keyword 'ID' has to be specified, if an task ID should be specified.

```
Examples:      niceppc id 103 -10      ;The task with the ID 103 gets the
                                   ;NICE value -10
               niceppc 5                ;The curren task gets the NICE value 5
```

1.20 Demo-Programs

The WarpOS-archive contains a number of demo-programs that demonstrate the capabilities of the PPC-processor and of WarpOS. The source is also supplied for most of the demos.

The 'exceptions' subdirectory in the demo-directory contains a couple of small programs that do nothing but crash. The purpose of these programs is the demonstration of the WarpOS-crash-requester.

The following demo-programs are available:

```
multitasking
tabletennis
semcorrupt
pixelOmania
cybermand
cyberpi
voxelspace
landscape
```

The 'multitasking' demo demonstrates the WarpOS-multitasking. It is started without parameters from the shell. This program generates 9 further PPC-tasks right after it has been started. Each of these tasks then opens a window an continously prints a text to it.

After that you will be asked to press CTRL-C in the shell-window. As soon as this happens the main task sends a signal to all of the 9 subtasks which then confirm the signal and terminate. After a short pause the program terminates. The windows must be closed manually.

Background information on this demo: These 10 tasks are no the only ones running - there are also 10 68K-tasks that were created as mirror-tasks to the PPC-tasks. The 68K-tasks do things like executing OS-functions that are not available in native PPC form (eg.g open window, print text). This program further demonstrates that the PPC can wait for the control-signals on its own although those were sent by the AMIGA-OS.

The subtasks inherit the priority of the mother-task. This means that if somebody changes the shell priority there is no danger of a deadlock occurring as would be the case if the subtasks had a predefined priority.

The 'tabletennis' demo-program demonstrates the communication abilities of WarpOS. The program calls a PPC-function which in turn calls a 68K-function, then another PPC-call, etc. In the beginnings some calls are passed back and forth and finally all calls are terminated by return jumps. This is repeated 10 times. The shell window always displays information on what is currently going on.

The 'semcorrupt' demo-program demonstrates a WarpOS-system message. Currently there are only a few occasions when a system requester appears. 'semcorrupt' allocates a semaphore and then frees it once too often. This causes such a system message to appear.

The demo 'pixelOmania' demonstrates the performance of the context switches. It opens a small window and draws pixels with alternating colors. This demo is a pure PPC demo (except for the startup code) which calls the AMIGA-OS system functions intensively. In the main loop, four system functions are called (GetMsg, ReplyMsg, SetAPen, WritePixel). At the end of the demo, the time elapsed during the demo is printed out. The demo 'pixelOmania' is suitable to compare the context switch performance between the powerpc.library V7 (based on the hardware setup of the ppc.library) and the powerpc.library V8+.

The 'cybermand' demo-program is a realtime Mandelbrot-program. It runs on a PAL-screen on ECS/AGA machines as well as on a graphics card using cybergraphics. The core routine of the Mandelbrot-program is written in highly optimized PPC-assembler-code and is completely adapted to the floating-point capabilities of the PowerPC. The PowerPC enables you to actually "dive" into the Mandelbrot-set in a smooth motion.

While the demo is running you can navigate freely within the Mandelbrot-set by moving the mouse. The left mouse button zooms into the set while the right button zooms out. The following keys are used by the program:

```
ESC      : quit the demo
SPC      : shows the area fullscreen with highest iteration depth
RETURN   : returns to realtime-mode after SPC was pressed
F1       : switches between 68K and PPC processor
Cursor keys : moving the mandelbrot area
Shift right : accelerating movements
Shift left  : Zoom in
Alt left    : Zoom out
```

'cybermand' has a few CLI-parameters:

```
WI=WINDOW      : The demo runs in a window on the workbench
W=WIDTH        : The width of the area to be displayed. The smaller this
```

value is, the faster the animation will be.

H=HEIGHT : The height of the area to be displayed. The smaller this value is, the faster the animation will be.

X=MAXSIZE : Realtime-mode on full screen

I=ITERATIONS : Number of iterations per pixel (standard = 100)

F=FULLSCREEN : Number of iterations in fullscreen-mode (if the SPC-key is pressed during the demo).

Z=ZOOMSPEED : A number between 1 and 90 that specifies the zoom speed.

M=MOVESPEED : A number between 1 and 5 that specifies the moving speed.

S=SLOWDRAW : SetAPen()/WritePixel() are used for drawing instead of a C2P-algorithm (only in PAL-mode).

C=COLOR : A number between 1 and 5 which is used to select the color.

SP=SINGLE : Single precision floating-point-operations are used.

P=PAL : The display is done on a PAL-screen even if a graphics card is present.

NM=NOMOUSE : Disables mouse control

Background information on 'cybermand': This demo is 100% system-compliant! It is a demonstration of what performance can be achieved without resorting to evil 'hacks'. It should also show the way to take in order to develop super-fast programs on the PowerAMIGA.

The main loop consists of a 68K-part that evaluates the window-messages and draws the graphics, and a PPC-part that does the actual calculations. Two PPC-calls are performed per iteration. The extremely fast WarpOS-communication leads to the enormous speed even in the smallest of resolutions.

'Cybermand' also works with powerpc.library V7. See Preface.

'Cyberpi' is a demo-program that calculates the number PI with any desired decimal precision. It has two parameters:

DECIMALS/N : Number of decimals to calculate after the decimal point.

M68K/S : Lets CyberPi run on the 68K processor.

After the program terminates the time it took to run is displayed.

The algorithm used in this demo is based on a formula that in turn is based on taylor series for the arcus tangent. It is definitely one of the fastest algorithms and even the pure 68K-version (which is not present here) is able to outperform alternative Pi-programs. The following times were measured in a test where 100000 decimal places were calculated:

68040/25 : about 30 min

68060/50 : about 15 min

PPC603E/150 : about 9 min

You can see that the algorithm is ideal for the 68040 because it is the only processor that has 64-bit-division.

The 'voxelspace' demo program is most likely to be one of the most interesting and at the same time breathtaking demos for the PowerPC. It is a realtime-3D-demo that is based on the voxelspace-technique that became popular with the well-known PC-game 'Comanche'. For the first time there is voxelspace-technique on the AMIGA - at an amazing speed only seen on competing

systems until this date.

'voxelspace' runs in PAL-mode (AGA) as well as on a graphics card using cybergraphics. When using cybergraphics you can choose a screen mode to use. The smaller the resolution - the faster the demo. Resolutions of about 320*256 yield good results.

As soon as the demo has been started, the landscape appears and at the upper left there is an information bar. The demo can be controlled using a mouse or the keyboard.

Mouse control:

mouse movement left/right	: horizontal rotation
mouse movement up/down	: climb/sink (only if manual climbing is switched on)
left mouse button	: accelerate
right mouse button	: decelerate

Keyboard:

cursor left/right	: horizontal rotation
cursor up	: accelerate
cursor down	: decelerate
Ctrl left	: climb (if manual climbing is on)
Alt links	: sink (if manual climbing is on)

By holding down the Shift-key, some actions can be performed faster.

If the TurboGfx- or CGFXPlus-options are switched on (see further below) crashes may occur when a mouse button is pressed. In this case it is advised to deactivate all commodity-programs running in the system (you can of course find out which program causes the problems and only switch that one off). Blanker programs are known to often cause this kind of problems.

The information bar at the top left displays the following information:

Freq	: The current frame rate in Hz. The higher, the smoother.
CPU	: The active CPU. The demo supports the 68K- as well as the PPC-processor.
MMU	: Signifies the MMU-status. 'Standard' means a normal MMU-setup, 'Turbo' means a specially optimized MMU-Setup.
Res	: The currently selected resolution.
Gfx	: The selected graphics mode:
	PAL : Display in PAL-mode
	CYBERGFX : Display on Gfxboard (system compliant)
	CYBERGFX+ : Display on Gfxboard (system compliant) with activated multi-buffering.
	TURBOGFX : Specially optimized display on

gfxboard (direct rendering into video memory).

Columns : The number of columns drawn on the screen. The higher this number, the more detailed the graphics. If this number is identical to the resolution width, the maximum number has been reached.

PixDepth : Shows how far you can see into the distance. The higher this value is, the slower the demo runs.

Approx : Shows if a special approximation-algorithm is used or not.

While the demo is running the following keys are used:

ESC : quit demo

SPC : show/hide info-display

F1 : Switch between 68K- and PPC-CPU

F2 : Switch between standard- and turbo-MMU-setup (Only for 68K and only if MMUENABLE or WARP were specified as parameters)

F3 : Set column width to 1

F4 : Set column width to 2

F5 : Set column width to 4

F6 : Decrease viewing distance (Shift is supported)

F7 : Increase viewing distance (Shift is supported)

F8 : Toggle approximation-algorithm

F9 : Toggle manual climbing
Usually the landscape height is adjusted automatically. If manual climbing is switched on, you can also fly high above the landscape.

F10 : Doubles/Halves the pixel depth in the third dimension

The following CLI-parameters are supported:

K=KEYS/S : Displays the list of used keys and their purpose

W=WARP/S : Maximum performance (TURBOGFX, MMUENABLE and TURBOPPC options are automatically switched on)

M=MMUENABLE/S : A more optimal MMU-setup for the 68K-processor is activated and switched on. All address translations are redirected to the transparent translation registers. Only for 68040/68060. This parameter should be considered a 'hack' and be treated with care.

T=TURBOPPC/S : A more optimal MMU-setup for the PPC-processor is activated and switched on. The memory areas for the map and the sky are marked as 'noncachable' using 'AllocVecPP' and assigned to a BAT-register. This option is NOT A HACK! It is a feature of the WarpOS operating system and there is no reason not to use this option (unless you are low on memory).

P=PAL/S : The demo runs in PAL-mode even if a gfxcard is available.

T=TURBOGFX/S : The landscape is generated directly in the video-memory instead of copying it there from FAST-RAM using a system-function. This access is

performed using the lock/unlock mechanisms of the cybergraphics.library and can therefore be considered as system-compliant (see also MODE2- and NOLOCK-options)

Since V1.6 this demo uses the OS3.0 compatible multibuffering, if Picasso96 is installed.

C=CGFXPLUS/S : The demo runs on the graphics board in system-compliant mode but in addition the multibuffering is switched on internally. This does not result in any performance increase compared to the conventional method but is very interesting for graphics board-programming (further details on this can be found in the 'GameDev.guide' document).

WI=WINDOW/S : The demo runs in a window on the workbench.

M=MODE2/S : The address of the bitmap is determined using an alternative method (through LockBitMapTagList). Only for TurboGFX. Should be tried if graphics-errors occur.

NL=NOLOCK/S : Disables the internal locking of the bitmaps (only when TURBOGFX enabled and MODE2 disabled). Should only be tried, if problems occur.

The remaining parameters are only for playing around - they change some voxelspace-specific parameters.

Background information on 'voxelspace': If no parameters are specified this program runs 100% system-compliant! If the additional TURBOPPC parameter is specified the program still runs 100% system-compliant but already at a very high speed. TURBOGFX can be used to squeeze even the last bit of speed out of the PPC.

Voxelspace is supposed to set the marching direction for a new generation of PPC-optimized games. It proves unambiguously and impressively that it is indeed possible to develop system-compliant games that run at a very high speed. This should also be an appeal to programmers to make their games system-compliant in the future and also offer alternative 'hacks' as additional options in the future. The user can then decide for himself if he wants the game to be 100% system-compliant or if he wants more speed.

Further thoughts and tips on this topic can be found in the 'GameDev.guide' document.

The internal organisation of 'voxelspace': Similar to 'cybermand', the demo consists of a 68K-part that takes care of the message-handling and a PPC-part that does the calculations. The PPC-part is called once per iteration. The PPC-part has again been written in highly optimized assembler-code that is adapted to the internal structure of the PPC-processor. The PPC-algorithm even contains mixed Integer/FP-areas that optimally exploit the internal execution units. Furthermore, all 32 integer-registers of the PPC are completely used by the program.

The demo 'landscape' is a fractal based landscape generator which supports both the 68K and the PPC processor, as well as CyberGFX and PAL. The demo must be started from the directory where the executable 'landscape' is located, so that the palette file 'colortable.bin' is found.

After the start, 'landscape' calculates the landscape, opens a screen and displays the graphics. If the landscape is bigger than the visible area of the screen, you can scroll the picture using the mouse. If there is too few display memory available, the landscape is scaled and a screen of smaller size is opened. The program terminates after clicking the left mouse button.

The demo supports the following CLI parameters:

M=M68K/S	: The landscape is calculated using the 68K processor.
P=PAL/S	: The landscape is displayed on a PAL screen, even if a graphics board exists.
WI=WINDOW/S	: The demo runs in a window on the workbench.
MS=MAPSIZE/N	: The size of the landscape to be calculated. This value must be specified as exponent to the base 2, so a landscape of the size 1024*1024 has to be specified using MAPSIZE 10 (because $1 \ll 10 = 1024$)
H/N	: A fractal parameter which defines the characteristics of the landscape. Possible values are 0-15.
I=ITERATIONS/N	: Defines how much sub landscapes are calculated. A landscape can be divided into 4 sub landscapes which are calculated independently (although the borders fit together). In this way the real size of the landscape increases and the resolution decreases with the number of iterations.
V=VARIATION/N	: A degree of the color variation (between 0 and 255). If no variation is specified, every pixel height is assigned one defined color, otherwise the color can vary.
W=WATER/N	: The water coverage in percent of the total landscape area size.
NS=NOSHADOW/S	: Forbids the calculation of the shadow.
LD=LIGHTDIR/N	: A number between 0 and 3 which defines the direction of the light (one number for each direction).
LA=LIGHTANGLE/N	: A number between 0 and 90 which defines the angle of the light rays. 90 degree means vertical light rays and therefore no shadow.
C=CODE/N	: If no code or 0 is given, every landscape is based on another seed value for the random generator. If a number unequal 0 is given, a certain landscape is generated. Every landscape gets assigned a number which is printed out to the shell at the end of the program.
R=RANCOUNT/N	: The number of uniform random numbers which used to calculate one normal random number.

This demo is based on a fractal generator with quadratic elements. As random generator, the URN30 algorithm was chosen, which generates random numbers of high quality.

1.21 Developer Support

The WarpOS-archive contains all necessary documentation needed for optimal programming under WarpOS. Below you find a description of all the different

documents. Special attention is devoted to assembler-programming.

To effectively develop PPC-programs you of course need a developer environment such as StormC (C/C++) or StormPowerASM (assembler).

The include-files

There are some include-files that should be copied into the standard-include directory. These files are located in the 'powerpc' subdirectory of the include-directory.

The following assembler include-files are available:

powerpc/ppcmacros.i	: This file contains a lot of standard-macros which are indispensable for convenient PPC programming. Almost every PPC source code will include this file. These additional commands are explained extensively in the StormPowerASM-assembler documentation.
powerpc/powerpc.i	: This file contains macros and structures related to hardware- and communication-specific matters.
powerpc/listsPPC.i	: This contains some macros for list management.
powerpc/memoryPPC.i	: Describes structures and constants for the WarpOS memory management.
powerpc/tasksPPC.i	: Describes structures and constants for the WarpOS multitasking.
powerpc/semaphoresPPC.i	: Describes structures for the WarpOS-semaphores.
powerpc/portsPPC.i	: Describes structures for the WarpOS-messageports.
libraries/powerpc.i	: The include file which is used, when software is developed which must run under powerpc.library V7.

The following C include-files are available:

clib/powerpc_protos.h	: Contains the prototypes of the library-functions.
stormprotos/powerpc_sprotos.h	: Contains the Storm-specific extensions for the prototypes.
pragma/powerpc_lib.h	: Contains some pragmas for calling 68K library-functions.
libraries/powerpc.h	: The include-file that should be used when developing software that has to work with version V7 of the powerpc.library.
powerpc/powerpc.h	: The include-file that should be used when all features of the powerpc.library are to be used.
powerpc/memoryPPC.h	: The C counterpart to memoryPPC.i
powerpc/tasksPPC.h	: The C counterpart to tasksPPC.i
powerpc/semaphoresPPC.h	: The C counterpart to semaphoresPPC.i
powerpc/portsPPC.h	: The C counterpart to portsPPC.i

The LVO-file:

The 'powerpc_lib.i' file defines the library-offsets of the powerpc.library which are necessary in assembler in order to call a library-function. It should be copied to a directory with the assign 'LVO:' pointing to it in order to ensure a standard setup for all programmers.

The LVO-file also contains the name of the powerpc.library as a macro as well as macros for calling library-functions for the 68K as well as the PPC.

The documentation of the library in Autodocs-format:

The 'powerpc.doc' file contains a detailed description of all functions of the powerpc.library. Each description will be similar in structure to the others. The following subtitles can occur in a description:

NAME	: The name of the function plus a short description of the function as well as the required version number of powerpc.library.
CPU	: Defines what CPU the function was implemented for. (680x0 or PowerPC)
SYNOPSIS	: Defines the user interface of the function. The parameters as well as the passing registers are stated as well as a prototype-description for C which also describes the data type of each parameter.
FUNCTION	: A detailed description of the function.
INPUTS	: A description of the input parameters.
RESULT	: A description of the return value.
NOTES	: Important notes regarding the use of the function.
SEE ALSO	: A reference to other descriptions that also contain information on the same topic.

In addition to the developer docs there are also some assembler source codes that can serve as examples. The source codes can be assembled using the StormPowerASM-assembler without using any special parameters.

In order to develop efficient PPC-programs in assembler, some knowledge of the PowerOpen-Standard is necessary as every PPC-function must be called according to PowerOpen-conventions. Of course, the same applies to the library-functions of powerpc.library.. Some information on this topic can be gained from the StormPowerASM documentation.

1.22 Frequently Asked Questions

This page is reserved for frequently asked questions. This also includes questions by WarpOS users that have an answer that is of public interest.

As many people usually have the same problems, this can save a lot work and hassle.

Currently this section is empty as the very first version of WarpOS has just been released. Do not fear - questions will definitely arise!

1.23 Thanks

First of all, thanks must go to 'Haage & Partner' and all of their employees for their simply outstanding support. Their great support and the trust they placed in me enabled me to lead this project to such a high quality.

Further thanks go to:

Michael Rock

Author of the WarpOS-memory management and FP-emulation algorithm.

Frank Wille

frank@phoenix.owl.de

Author of the PhxAss-assembler which was used to develop the 68K-part of the powerpc.library.

1.24 Literature Index

Here is a list of all literature that was used while developing WarpOS.

Programmer's Reference Manual (Motorola)

THE reference manual for 68xxx-commands.

Amiga Intern (Data Becker)

PowerPC Microprocessor family: The programming environments (Motorola /IBM)

This manual is a must for all aspiring PowerPC-assembler-programmers

It is similar in structure to the Programmer's Reference for the 68xxx-processors.

PowerPC 603 User's Manual (Motorola /IBM)

Recommended. Contains a detailed description of all commands like the 'Programming Environments' as well as a directory of all extended mnemonics.

PowerPC 603E User's Manual (Motorola /IBM)

PowerPC 604 User's Manual (Motorola /IBM)

Optimizing PowerPC Code (Addison Wesley)

Recommended if information on stack frames, programming-guides and optimizations is desired.

1.25 Support

We are always happy to receive suggestions, ideas, criticism, reactions, etc. If any errors occur we are very grateful if you inform us of them.

Please document all errors using the debugging-option ('setdb' tool). It is very important that the error is reproducible. Please send the debug-output to us. If any crashes occur, we would of course like to know the content of the crash-requester. Finally, please tell us which version of the powerpc.library

you use.

How to contact us:

regular mail:

HAAGE&PARTNER GmbH
Mainzer Straße 10a
61191 Rosbach v.d.H
Germany
Tel: ++49/ (0) 6007/930050
Fax: ++49/ (0) 6007/7543

eMail:

warpup@haage-partner.com