4P-3

# CPX – An Operating System Kernel for the CPC

Paul Spee *, Mitsuhisa Sato *, Norihiro Fukazawa *, Eiichi Goto * **

* Research Development Corporation of Japan (JRDC)
** Riken, Institute of Physical and Chemical Research, and
University of Tokyo, Department of Information Science

## 1. Introduction

This paper describes the basic structure and functionality of the CPX kernel. The CPX kernel is a message passing, object based kernel for the Cyclic Pipeline Computer (CPC), a computer which implements a tightly coupled MIMD multiprocessor architecture by timesharing the processor and the main memory among a fixed number of instruction streams.

## 2. Kernel functionality

The following goals were put forward in designing the CPX kernel:

- efficient use of the shared memory multiprocessor structure of the CPC,
- portability to a wide range of (parallel) architectures, and
- extensibility at 'user' level.

### 2.1 Object oriented programming model

The CPX kernel is an object oriented kernel. It implements several kernel defined object. Message passing is used to implement the object oriented programming model. Sending a message to an object represents an operation on that object. Various object oriented operating systems have been implemented. Most notably Hydra [WULF81], StarOS [JONES87], Eden [BLACK85] and most recently Mach.

### 2.2 Message passing primitives

In this section we describe the message passing primitives as implemented in the kernel.

*Communication channels*
A communication channel designates the source and destination of a message. The CPX kernel implements *ports* as a communication channel. A ports allows for multiple senders and a single receiver.

*Port descriptors and capabilities*
Port descriptors are handled analogous to the file descriptors in UNIX. Each task has access to a number of ports through port descriptors.
Associated with each port is the right to send messages to the port and the right to receive messages from the port. We call these rights the send and receive *capabilities*. A task may pass the port capability to other tasks.

*Send and receive primitives*
The CPX kernel supports *asynchronous* message passing with blocking/nonblocking send and receive. In case of blocking send/receive, a time-out parameter can be specified. Asynchronous message passing allows for a higher degree of concurrency than *synchronous* message passing.

*Messages*
A message consists of a header and a collection of typed values. The types are modeled after C and include basic types such as *void* (ignore type), *char*, *long*, *float* etc. Furthermore, it has composite types such as *pointer* to basic type, *range* of basic type, *array* of basic type. A special type is the *capability* type, which is used to pass port descriptors. Reasons to implement typed messages are:

- Messages of different type can be treated differently by the receiver.
- Representation of values can be converted when sending messages between different machines (e.g. MC68000 and VAX).
- The kernel can intercept messages with *range* types and *capability* types.

## 3. Kernel structure

To hide the hardware from the rest of the operating system, we define a virtual machine. The virtual machine consists of the virtual processor (vp), the virtual space (vs) and cache.

The kernel implements *virtual instructions*, which can be seen as extension of the processor instruction set, and *operations* on kernel defined objects (virtual processor, virtual space, and cache).

**Virtual processor**
The virtual processor is the virtualization of the real processor and includes the registers, processor status, stack pointer, and program counter. Its function is three fold:

- isolate nasty hardware details, such as interrupts, from the rest of the kernel,
- extend 'functionality' of the hardware by providing *virtual instructions*, and
- provide multi-programming, by providing a (fixed number of virtual processors, which are scheduled using pre-emptive scheduling.

CPX – An Operating System Kernel for the CPC
Paul Spee *, Mitsuhisa Sato *, Norihiro Fukazawa *, Eiichi Goto * **
* Research Development Corporation of Japan (JRDC)
** Riken, Institute of Physical and Chemical Research, and University of Tokyo, Department of Information Science

All operations related to ports and messages are implemented as virtual instructions; that is, they are implemented as system call traps. All operations on kernel objects are implemented as sending a message to the port representing that object.

| instruction | description |
|---|---|
| port_create | create new port in virtual space |
| port_delete | delete port |
| port_send | send message |
| port_receive | receive message |
| **operation** | **description** |
| vp_create | create new virtual processor |
| vp_delete | delete virtual processor |
| vp_will_suspend | vp is ready to be suspended |
| vp_wait | suspend virtual processor |
| vp_resume | resume virtual processor |
| vs_create | create new virtual space |
| vs_delete | delete virtual space |
| vs_allocate | allocate memory object in vs |
| vs_deallocate | deallocate address range |
| cache_create | create cache for memory object |
| cache_delete | delete cache |

**Virtual space**
Just like the virtual processor hides hardware details concerning the processor, the virtual space hides the details of the virtual memory implementation. The virtual space not only provides the address space for the virtual processors, but also holds the port capabilities created by or passed to that virtual space.

**Memory objects**
CPX takes its implementation of the virtual memory system from Mach [TEVANIAN87]. A memory object is an object which represents an amount of data. Data can be read or written by sending a read or write request to the memory object. Memory objects are mapped onto the virtual space.

The kernel provides a default memory object which represents zero filled memory. A default data manager handles the default memory object. User provided memory objects are managed by an user data manager. The purpose of a data manager is to support paging.

**Cache**
The virtual memory is used as a very large cache for the mapped memory objects. The function of the software cache is similar to that of a hardware cache. The most recently accessed pages are kept in the main memory. Other pages, which are not required, can be paged out. To manage the memory object and the cached pages, the virtual machine implements a cache object, which represents the cached pages (pages in physical memory) of the memory object. When a page fault occurs, the kernel looks up the faulted page in the cache object. If the page is not found, it sends a message to the memory object, requesting the data.

**4. User extensibility**

The port and the send/receive primitives provided by the kernel are used to implement objects. How user objects are implemented is not determined by the kernel, but is up to the user. Each single object could be implemented using a virtual processor. It also possible to implement an object server, which provides the unit of execution for the instances of the object type (super class) it defines.
By defining the kernel objects, the kernel does not constitute an operating system. It provides the frame, wheels and motor, but it does not provide a car. This allows us to implement operating systems on top of the kernel with different semantics. For example, the operating system could implement the single-threaded UNIX process or the multi-threaded Mach task.
The creation of a process could be defined as create new virtual space, allocate text segment, allocate data segment, create new virtual processor (including allocation of stack segment).

**5. Summary**

In this paper we presented the CPX kernel which provides the basic functionality for obtaining concurrency through asynchronous message passing.

We summarize the goals as stated in section 2 and their implementation.

- Allowing multiple virtual processors to execute in one virtual space reduces the overhead caused by context switching.
- Using the message passing model as concurrent programming model allows implementation on a wide range of parallel architectures.
- User extensibility is provided by the object oriented model (supported by message passing).

**References:**

[BLACK85]
Andrew P. Black, "Supporting Distributed Applications: Experience with Eden", *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, Orcas Island, Washington, 1-4 December 1985, pp. 181-193.

[JONES87]
Anita K. Jones, Robert J. Chansler Jr., Ivor Durham, Karsten Schwans, and Steven R. Vegdahl, "StarOS, a Multiprocessor Operating System for the Support of Task Forces", *Proceedings of the 7th Symposium on Operating System Principles*, December 1987, pp. 117-127.

[TEVANIAN87]
Avadis Tevanian, Jr., "Architecture-Independent Virtual Memory Management for Parallel and Distributed Environments: The Mach Approach", 1987.

[WULF81]
William A. Wulf, Roy Levin, and Samual P. Harbison, "HYDRA/C.mmp - An Experimental Computer System", McGraw-Hill, 1981.