| bcc | title | prefix/class-number.revision |
|---|---|---|
| | Processes | PROC/W- 36 |

| checked | authors | approval date | revision date |
|---|---|---|---|
| checked | Rainer Schulz | 12/4/69 | |
| approved | | classification: Working Paper | |
| | | distribution: Company Private | pages: 27 |

## ABSTRACT and CONTENTS

This document describes the manipulation of processes in the basic system.

CONTENTS PAGE NO.


| | |
|---|---|
| Introduction | 1 |
| Process Creation | 2 |
| Process Initialization | 5 |
| Process Activation | 7 |
| Making Processes Dormant | 8 |
| Communication with Processes | 9 |
| Processes and Terminals | 10 |
| Linking | 12 |
| Destroying Processes | 13 |
| Non-interactive Processes | 14 |
| Monitor Calls: | |
| Open Process | 16 |
| Destroy Process | 17 |
| Make Process Dormant | 18 |
| Transfer a Terminal | 19 |
| Activate Process | 20 |
| Link to Process | 21 |
| Break Links | 22 |
| Copy PMT | 23 |
| Make Page | 25 |
| Initialize Page | 27 |

## Introduction

This document deals with the monitor calls for creating and destroying processes and for making them active or dormant. It also describes the implementation of those operations.

A process is defined by its context block, which is a page containing all the information required to run the process: its state, map, open files and other objects maintained by the monitor, call stack etc. Every process also has an entry in some MIB which points to the context block. Like any MIB entry, this one includes a name; it is therefore possible to refer to any process symbolically. It also includes all the protection mechanisms included in any MIB entry; access to the process can therefore be controlled in the same way as access to a file.

When a process is not being used for a long period of time, the MIB and context block are all that is required to keep it in existence. The context block normally resides on the disk in this case, and the process is said to be dormant. The only operations which can be done on a dormant process are

1) destroy it
2) make it active
3) initialize it

These operations are only legal for a dormant process.

Creating a Process

To create a new process, the monitor call "create new entry with name n as type t" is used. This call is described in SCBFS/W-9 and MIFS/W-1Ø. Any restrictions on these calls also apply to the creation of processes.

A dormant process is created. It has no resources. Space for its context block is charged to the MIB in which it is created. The context block is put in the active page set of the creating process so that someone will be paying for keeping the context block on the drum while the monitor initializes it.

The open file table, sub-process table, call stack and PMT of the process are cleared. The context block is set up so that the process will start execution at an initialization point in the monitor. The context block is initialized with all the necessary PMT entries, the map, etc. necessary to run the monitor. After this call the process has been created as far as the basic system is concerned. If someone activates the process at this time, the process will automatically become dormant again, since no sub-processes have been created in the process yet, and the monitor doesn't interface to the user. The process is now in the following state:

a) dormant - not open for modification

b) not initialized

When a process is made active, it acquires an additional piece of machinery called a PRT (process resident table) entry. This 12-word object contains the information required to allow the rest of the system to communicate with the process: schedule it, swap it in, send it wakeups, report disk errors, send it alarm-clock signals and keep track of its status while all these things are going on. As is suggested by its name, this table is resident in core. Tables maintained by the CHIO and AMC can contain pointers to the PRT entry of a process.

The context block for an active process normally resides on the drum.

A process can be in several states:

1) dormant
2) dormant - open for modification
3) being made dormant
4) active
5) being activated
6) being destroyed

All of these states are discussed in subsequent chapters. States 3, 5 and 6 are transitory states. The following sequences of states are possible (assuming one starts with a dormant process).

a) 1 → 2 → 1
b) 1 → 5 → 4 → 3 → 1

In addition to these states there is an attribute called "initialized", which is also described later on.

Initializing Processes

After a process has been created, it has to be initialized so it can really run and execute user code.

The process has to be "opened for modification" before any initialization can take place. If this call is successful an OFT entry for the process is made in the calling process. The following conditions have to be satisfied for the call to succeed

1) the process has to be dormant
2) it cannot be "open for modification" by any other process
3) the caller has write access to the process
4) process cannot have been initialized previously

The process stays "open" in this way for not more than 15 minutes.

The process gets closed (the OFT entry is removed) if:

1) the process is "closed" by the user
2) the process has been "open for modification" for more than 15 minutes

At the time the process is closed its state goes from "not-initialized" to initialized. After opening the process several calls can be made on the monitor to initialize PMT, SPT and the utility area of the context block. In order to initialize the SPT and PMT entries, the "attach"

function in the utility should handle the case of initializing new processes. The implication of these monitor calls is that any "attach" file created by SPL can be started up as the father of all fathers (controlling sub-process) in the new process.

This is true whether the first sub-process runs in the utility ring or in the user ring.

Activating a Process

In order to activate a process, it has to be

a) dormant - not open for modification

A process is activated by transferring resources to it. This operation is simply a resource transferring operation where one of the targets is a dormant process. The resources being transferred must include at least one process and at least n cycles/second of CPU; they must be valid for at least m minutes into the future. The resources may also include a terminal.

When a process is activated, a PRT entry is set up for it, its context block is brought from disk to drum and it is woken up. It will have been blocked in the monitor, either at the initialization entry point mentioned above, or in the 'go dormant' (GD) routine. In the latter case, it puts the context block in its APS and returns control to whatever routine called GD.

In the former case, the initialization code in the monitor calls the first sub-process at a special entry point. As soon as the process is activated, its state goes from "dormant" to "active". If there is no first sub-process, the process becomes dormant again.

Making a Process Dormant

An active process can be made dormant in two ways. The process can call on the monitor to make itself dormant, or another process can send an interrupt to the process, requesting that it make itself dormant. The result is the same in either case. When the monitor receives the "go dormant" request it:

1) removes all pages from the active page set.

2) breaks links
3) returns the controlled terminal (if there is one) to the system and disconnects the line
4) returns whatever resources may be left to the MIB that the process belongs to (an FRA with the same name as the process)
5) releases the PRT entry

As soon as all this is done, the process is said to be dormant.

A process may also automatically become dormant if its resources have expired and no new resources have been put in the appropriate FRA of the process's MIB.

Communicating with Processes

It is possible to send wakeups to active processes (see IWS/W-11). To do this it is necessary to open the process. This is done exactly like opening a file (using an OFT entry). There is then an operation to send a wakeup. It takes the process number and a wakeup bit word as parameters. The bit word gets merged into PIW and the process gets woken up. Of course it fails if the PRT entry has disappeared. This "opening" of a process is not to be confused with "opening for modification." In this case the state of the process is in no way affected.

Processes and Terminals

The M1 system lacks any comprehensive facilities for controlling access to terminals. The following minimal scheme will be used.

Each line has in the CHIOs tables a process identification in the form of a PRT index; we will call this process the presumptive owner of the line (terminal). Processes are of two kinds with respect to lines, normal and special. A special process may access any line; specialness is acquired through the use of the Complete Teletype Access capability.

A normal process has an attribute called the controlled terminal, which is either null or is a line number. The process may access its controlled terminal provided that it is the presumptive owner; if this is the case, we call it the owner. Thus a normal process may access zero or one terminals, and a terminal may be accessed by zero or one normal processes.

A process A which owns a terminal may transfer ownership to another process B (which A must have opened with suitable access), provided that the calling subprocess has the capability to do so. The terminal becomes the controlled terminal of B, and B becomes its presumptive owner. A process

which owns a terminal may also give it up to the system. If this happens, the process becomes non-interactive.

The (special) listener process will transfer each terminal which dials in to a newly created utility process, provided that the necessary resources are available.

Linking

In order to link to another process, the process to be linked to has to be opened with proper access. Then any kind of linking can take place, provided the opened process owns a terminal and the target line has its status set to accept linking and is not already linked. If process A links to process B, then process A is said to have attached line B. Regardless of the kind of linking being done, process B also has line A attached. The access to an attached line depends on the type of link established. These types are documented in the CHIO specification FOO/S-2. If process A transfers its controlled line to process C, the link and the attached terminal are also transferred to process C.

## Destroying a Process

A dormant process with no private memory can be destroyed. This is done by simply releasing the MIB entry and the space for the context block. To make sure that the private memory goes away one operation is provided:

1) A monitor operation to scan the context block of another process and release the private memory.

Non-interactive-Processes

The facilities described here should be sufficient to permit the utility to implement a satisfactory scheme for non-interactive execution. The trickiest part is a mechanism for activating a process at some long-distant time.

# APPENDIX A

Brief list of Monitor Calls concerning the creation of processes and terminal control.

1) create process
2) open process
3) open process for modification
4) set PMT in opened process
5) set SPT in opened process
6) initialize new process CB (or created page)
7) close process (setting control lock to zero (Ø))
8) activate process
9) make process dormant
10) destroy process
11) link to process
12) break links
13) transfer terminal (either to system or to another process). Terminal has to be controlled or the caller has to be the presumptive owner.
14) create private memory page in new process

Open Process (Same as Open File)

Function OPPRM(N,RO,K), FRETURN, MONITOR ← 19;

SRETURN: OFT index

FRETURN: a) no process with name N

b) no write access to process, and/or no WFI capability (if RO = 1)

c) OFT full

d) process has been initialized (if RO = 1)

e) process not dormant (if RO = 1)

f) one process already open for modification (if RO = 1)

g) no access to process (if RO = Ø)

RO: if RO = 1, then the process is opened for modification, otherwise it is only opened for sending interrupts or linking.

K: OFT index for getting access.

Destroy Process (Same as "Delete entry N" as described in file system)

Function DESTPR(N,K), FRETURN, MONITOR ← 5 ;

SRETURN: no parameter returned

FRETURN: 1) no owner access to process

2) process does not exist

3) process not dormant

Make Process Dormant

Function GO'DORM(N), FRETURN, MONITOR ← 13 4;

N = process name (same as file name in file system) If N[2] = Ø, then the running process that made the call is made dormant, otherwise an interrupt is sent to the addressed process, which will cause it to become dormant.

SRETURN: after process is activated again (if own process is made dormant), or after interrupt is sent to named process. No parameters are returned.

FRETURN: 1) process does not exist

2) no owner access to process

Transfer a Terminal

Function TRTERM(X,N,RES), FRETURN, MONITOR ←133;

X = -1 controlled teletype

X = n, where -1 $<$ n $\leq$ Max line number

N = process name (same as file name in file system).

If N[2] = Ø, then terminal gets transferred to system. This also causes the terminal to be disconnected from the system. If RES = Ø, no resources get transferred for the terminal; if RES = 1, resources do get transferred to the target process.

SRETURN: none

FRETURN:
1) process does not exist
2) process dormant
3) process interactive
4) no owner access to process
5) calling sub-process does not have terminal transfer capability
6) terminal is not controlled or caller is not presumptive owner of terminal
7) expiration of resources of target process exceeds those of originating process

Activate Process

Function ACTPR(N,X,RES,K), FRETURN, MONITOR ← 132;

N is the name of the process, X is the terminal to be transferred. X and RES have the same meaning as in "transfer terminal." The same restrictions on the transfer of terminals also apply in this call, except that if X = -2, no terminal is to be transferred. The resources to be allocated to the process are taken from a fixed resource allocation block (FRA) in the MIB of the process. The name of the FRA is expected to be identical to the name of the process, except the type is "NRES."

SRETURN: PRT number

FRETURN:
1) process does not exist
2) no resources available
3) no owner access to resources or process
4) process not dormant
5) old version of process (cannot be run under current system)
6) calling sub-process does not have terminal transfer capability
7) terminal is not controlled or caller is not presumptive owner of terminal

Link to Process

Function LINKPR(N,T), FRETURN, MONITOR ← ;

T = type of linking to be done

T = 1) normal link

2) CCP mode

3) advise mode

SRETURN: line number linked to

FRETURN: 1) no read access to process (if normal linking)
or no write access if CCP mode, or advise mode

2) process is not active

3) process refuses linking

4) process already linked

Break Links

Function BRKLNK(), MONITOR ← ;

SRETURN: no parameters returned

FRETURN: function never fails

COPY'PMT - Copy a PMT Entry from One Process to Another

Declaration:

FUNCTION COPY'PMT(OFTX,PMTX), FRETURN, MONITOR ← 13Ø;

Success Return:

RETURN PMTX'; where PMTX' is the index of the PMT entry in process OFT[OFTX] into which PMT[PMTX] was copied.

Failure Returns:

(1) FRETURN('OFI', 36) unless 1 ≤ OFTX ≤ OFTL.

(2) FRETURN('OFA', 13) unless the calling sub-process has access to OFT[OFTX].

(3) FRETURN('ONP', 69) unless OFT[OFTX] is a process open for modification.

(4) FRETURN('OFE', 12) if OFT[OFTX] is empty.

(5) FRETURN('PMI', 11Ø) unless 1 ≤ PMTX ≤ NPMTE.

(6) FRETURN('PMC', 121) unless PMT[PMTX] is controlled by the calling sub-process.

(7) FRETURN('PNF', 185) unless PMT[PMTX] is a file page.

(8) FRETURN('NSP', 183) if SPT[1] in process OFT[OFTX] is free.

(9) FRETURN('PMO', 184) if there are no free PMT entries in process OFT[OFTX].

Action:

A free PMT entry in process OFT[OFTX] is acquired for SPT[1] of that process. The contents of PMT[PMTX] in the current process are copied into this new PMT

entry in the process being modified. The PMT index, PMTX', of the entry acquired is returned to the calling sub-process of the current process.

MAKE'PAGE - Create a Private Memory Page in a Process Open for Modification. (Same as create page for file)

Declaration:

FUNCTION MAKE'PAGE(OFTX,PMTX), FRETURN, MONITOR ← 24;

Success Return:

RETURN PMTX'; where PMTX' is the index of the PMT entry in process OFT[OFTX] in which the page is created.

Failure Returns:

(1) FRETURN('OFI', 36) unless $1 \leq$ OFTX $\leq$ OFTL.

(2) FRETURN('OFA', 13) unless the calling sub-process has access to OFT[OFTX].

(3) FRETURN('ONP', 69) unless OFT[OFTX] is a process open for modification.

(4) FRETURN('OFE', 12) if OFT[OFTX] is empty.

(5) FRETURN('PMI', 11Ø) unless $1 \leq$ PMTX $\leq$ NPMTE.

(6) FRETURN('PMC', 121) unless PMT[PMTX] is controlled by the calling sub-process.

(7) FRETURN('PMF', 122) unless PMT[PMTX] is empty.

(8) FRETURN('PMO', 184) if there are no free PMT entries in process OFT[OFTX].

(9) FRETURN('NSP', 183) if SPT[1] in process OFT[OFTX] is free.

(10) FRETURN('KSE', 186) if process OFT[OFTX] has no disk space available for the new page.

Action:

A free PMT entry in process OFT[OFTX] is acquired for SPT[1] of that process. A private memory page is created in this PMT entry (call it PMT[PMTX']). This page is acquired for process OFT[OFTX] in the sense that its Unique Name is derived from that of that process' Context Block and its disk space is charged against the disk allowance of the user to whom that process belongs.

The contents of PMT[PMTX'] is copied into PMT[PMTX] in the calling process. The FP flag in PMT[PMTX] is set. PMTX' is returned to the caller.

INIT'PAGE - Initialize utility area of CB or private memory page in new process.

Declaration:

```
FUNCTION INIT'PAGE (OFTX,PMTX', ST, NW, INTEGER ARRAY
                                    PG), FRETURN, MONITOR ← 135;
```

Success Return:

```
RETURN;
```

Failure Returns:

(1) FRETURN ('OFI', 36) unless $1 \leq OPTX \leq OFTL$

(2) FRETURN ('OFA', 13) unless the calling sub-process has access to OFT[OFTX]

(3) FRETURN ('OFE', 12) if OFT[OFTX] is empty

(4) FRETURN ('ONP', 69) if OFT[OFTX] is not a process open for modification

(5) FRETURN ('XXX', XXX) unless PMTX' is a private memory page

(6) FRETURN ('EWC', 50) excessive word count given

Action:

The private memory page PMTX' in process OFT[OFTX] is initialized with the contents of array PG. ST gives the starting location within page PMTX' and NW provides the number of words to be transferred. The data to be transferred is always taken out of PG[Ø] and the following locations. PMTX' determines whether the utility area of the CB(PMTX'=1) or another private memory page gets initialized.