MASENO UNIVERSITY

SCHOOL OF COMPUTING & INFORMATICS

CIT 110/CIM 112/CIS 112: Operating Systems/Platform Technologies I

Date: Wednesday, February 5, 2020

Week 4: Topic 2: Process Management

Is the ensemble/collective of activities of planning and monitoring the performance of a process. In multiprogramming systems the OS must allocate resources to processes, enable processes to share and exchange information, protect the resources of each process from other processes and enable synchronization among processes. To meet these requirements, the OS must maintain ***a data structure*** for each process, which describes the state and resource ownership of that process, and which enables the OS to exert control over each process

What is a process?

A process is a sequential program in execution. The components of a process are the following:

i. The object program to be executed (called the *program text* in UNIX)
ii. The *data* on which the program will execute (obtained from a file or interactively from the process's user)
iii. *Resources* required by the program (for example, files containing requisite information)
iv. The *status* of the process's execution

Two concepts emerge:

- Uni-programming: - case whereby a system allows one process at a time.
- Multi-programming: - system that allows more than one process, multiple processes at a time.

A process comes into being or is created in response to a user command to the OS. Processes may also be created by other processes e.g. in response to exception conditions such as errors or interrupts.

DESCRIPTION OF THE PROCESS MODEL / STATES

PROCESS STATES

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. Process state determines the effect of the instructions i.e. everything that can affect or be affected by the process. It usually includes code, particular data values, open files, registers, memory, signal management information etc. We can characterize the behavior of an individual process by listing the sequence of instruction that execute for that process. Such listing is calling the trace of processes

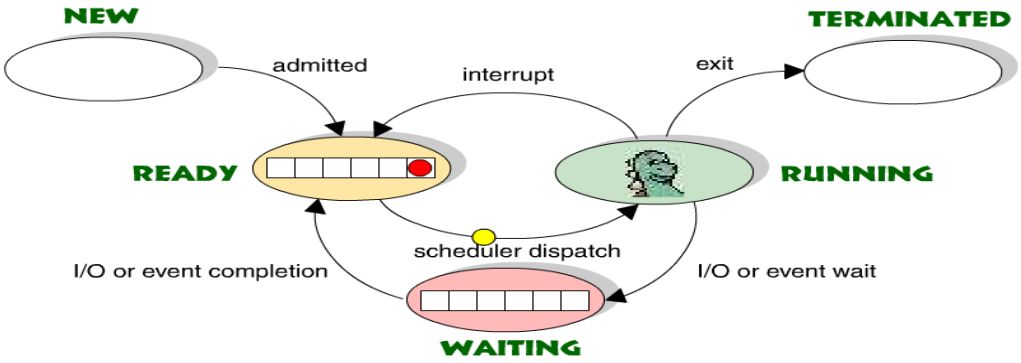
Each process may be in one of the following states

i. New: The process is being created

ii. **Running**: The process is being executed i.e. actually using the CPU at that instant
iii. **Waiting/blocked**: The process is waiting for some event to occur (e.g., waiting for I/O completion) such as completion of another process that provides the first process with necessary data, for a synchronistic signal from another process, I/O or timer interrupt etc.
iv. **Ready**: The process is waiting to be assigned to a processor i.e. It can execute as soon as CPU is allocated to it.
v. **Terminated**: The process has finished execution

A transition from one process state to another is triggered by various conditions as interrupts and user instructions to the OS. Execution of a program involves creating & running to completion a set of programs which require varying amounts of CPU, I/O and memory resources.

Figure: Process Life Cycle



Process Control Block (PCB) / Task Control Block.

The OS must know specific information about processes in order to manage and control them. Also to implement the process model, the OS maintains a table (an array of structures), called the process table, with one entry per process.

PCB information is usually grouped into two categories: Process State Information and Process Control

Information. Including these:

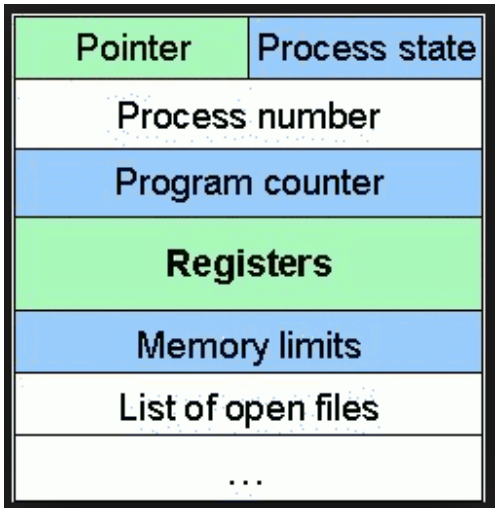


Figure 2: Process Control Block

i. Process state. The state may be new, ready, running, waiting, halted, and so on.
ii. Program counter. The counter indicates the address of the next instruction to be executed for this process.
iii. CPU registers. The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information.
iv. CPU-scheduling information. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
v. Memory-management information. This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the OS.
vi. Accounting information. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
vii. I/O status information. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

OS must make sure that processes don't interfere with each other, this means i. Making sure each gets a chance to run (scheduling).
ii. Making sure they don't modify each other's state (protection).

The dispatcher (short term scheduler) is the inner most portion of the OS that runs processes:

- Run processes for a while.
- Save state
- Load state of another process.
- Run it etc.

It only run processes but the decision on which process to run next is prioritized by a separate scheduler.

When a process is not running, its state must be saved in its ***process control block***. Items saved include:

i. Program counter

ii. Process status word (condition codes
etc.). iii. General purpose registers
iv. Floating - point registers etc.

When no longer needed, a process (but not the underlying program) can be deleted via the OS, which means that all record of the process is obliterated, and any resources currently allocated to it are released.

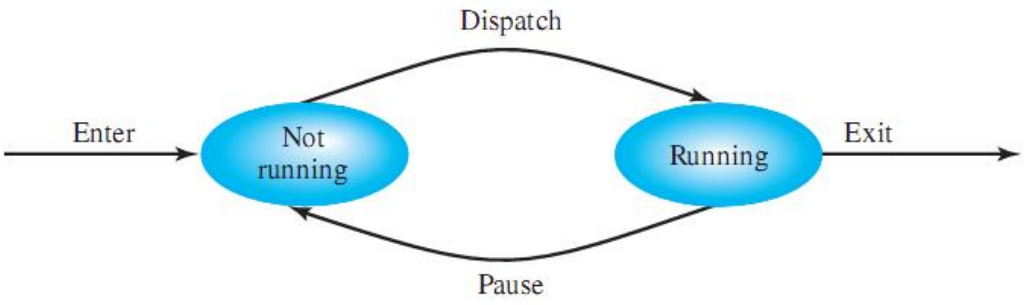
THERE ARE VARIOUS MODELS THAT CAN BE USED

i. Two State Process Model
ii. Three State Process Model
iii. Five State Model

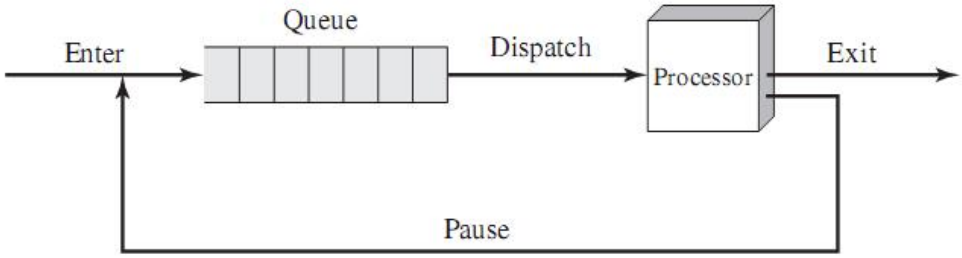
1. Two State Process Model

The principal responsibility of the OS is to control the execution of a process; this will include the determination of interleaving patters for execution and allocation of resources to processes. We can contrast the simplest model by observing that a process can either executed or not i.e. running or not running

Each process must be presented in some way so that the OS can keep track of it i.e. the process control block. Processes that are not running must be kept in some sort of a queue waiting their turn to execute.
There is a single queue in which each entry is a pointer to the PCB of a particular lock.



(a) State transition diagram



(b) Queuing diagram

ii. Three State Process Model

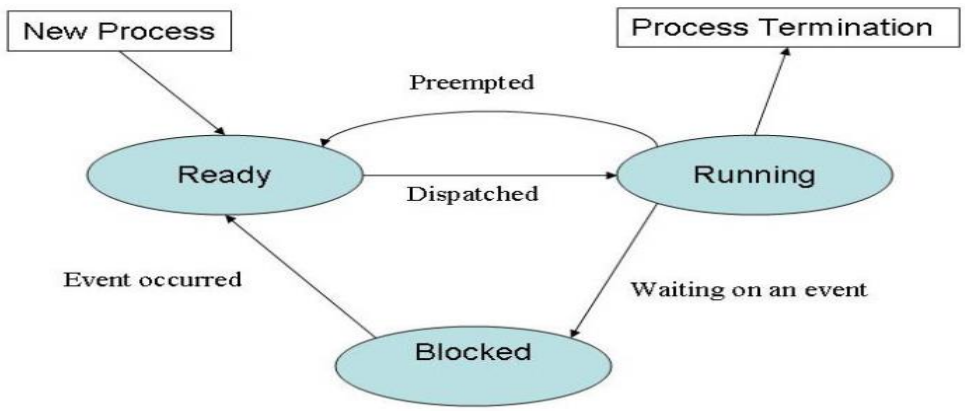


Figure 3: Three State Process Model

This state diagram is perhaps the most important in the entire study of Operating Systems, so it is important to understand what is happening in each state and how a process transitions from one state to another.

Here is what is happening at each transition.

1. Ready to Running This event is triggered by a kernel operation called the process scheduler or dispatcher, which selects a process in the Ready state and starts running it.
2. Running to Blocked This event would be triggered when a running process attempts to execute an IO operation such as a read or write or otherwise needs a resource which is not currently available. Accessing data from a disk takes several orders of magnitude longer than executing statements in the CPU, and so it makes sense for a process which is waiting for data from a disk to give up the CPU to another process while it is waiting.
3. Blocked to Ready A process is in the waiting state when it is waiting for an event to occur, so when this event occurs, it moves from Waiting to Ready. For example, a process which was waiting for an IO operation would be unblocked when the IO operation was completed and the data had been copied to or from user space.
4. Running to Ready A process moves from Running to Ready when the *Process Scheduler* determines that there is a process with a higher priority in the Ready queue. In many operating systems, if there are several runnable processes, each process is allocated a time slice or time quantum, and when this time slice ends, the running process is replaced by another process which is also ready to run, and the formerly running process is returned to the ready queue.
5. New to Ready When a new process is created, it is moved to the ready state.
6. Running to terminated There are two ways that a process can terminate. A normal termination occurs when the process reaches the end of main or calls the exit routine. An abnormal termination occurs when a process encounters an abnormal condition from which it cannot recover, such as a segmentation fault (memory access error) or when the user manually terminates the process.

The process of switching from one process to another is called *Context Switching*. If there are several runnable processes, all of them are loaded into memory. To execute a context switch, the kernel has to store the contents of all of the registers for the process which was formerly running, copy the register values for the new process into the registers, and start executing the next instruction of the new process. Thus, context switching can be done quickly.

This scenario is an oversimplification. The process scheduler is itself a process and so has to be swapped into the running state, although it is in the kernel, and is always in memory. Also, other kinds of interrupts may occur. Clock interrupts happen many times a second, and so the running process has to be preempted to handle the new interrupt.

iv. Five State Process Model

In this model two states have been added to the three-state model i.e. the ***new and exit*** state. The new state corresponds to a process that has just been defined e.g. a new user trying to log onto a time sharing system. In this instance, any tables needed to manage the process are allocated and built.

In the new state the OS has performed the necessary action to create the process but has not committed itself to the execution of the process i.e. the process is not in the main memory.

In exit state a process may exit due to two reasons

i. Termination when it reaches a natural completion point
ii. When its aborted due to an unrecoverable error or when another process with appropriate authority causes it to stop

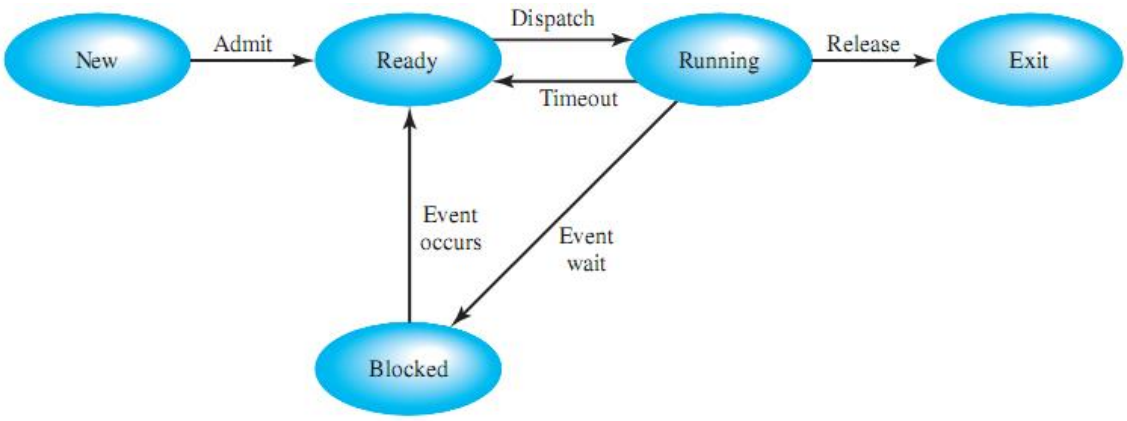


Figure 4: Five State Transition Diagram

i. **Running**: The process is currently being executed i.e. actually using the CPU at that instant
ii. **Ready**: The process is waiting to be assigned to a processor i.e. It can execute as soon as CPU is allocated to it.
iii. **Waiting/blocked**: The process is waiting for some event to occur (e.g., waiting for I/O completion) such as completion of another process that provides the first process with necessary data, for a synchronistic signal from another process, I/O or timer interrupt etc.
iv. **New**: The process has just been created but has not yet being admitted to the pool of executable processes by the OS i.e. the new process has not been loaded into the main memory although its PBC has been created.
v. **Terminated/exit**: The process has finished execution, or the process has been released from the pool of executable processes by the OS either because it halted or because it was aborted for some reasons.

In a five-state model the state transition are as follows

i. Null ----- New
A new process is created to execute a program
ii. New ---- Ready
The OS will move the process from new state to ready state when its prepared to take on an additional process.
iii. Ready ----- Running
When it's time to select a new process to run the OS chooses one of the processes in ready state
iv. Running ---- Exit
The currently running process is terminated by the OS if the process indicate that it has completed or its aborted.
v. Running ----- Ready
The common reason for this condition is that the running process has reached the maximum allowable time for uninterruptible execution
vi. Running ---- Blocked
A process is put into a blocked state if it requests something for which must wait e.g. when process communicate with each other, process may be blocked when its waiting for another process to provide an input or waiting for a message from another process

vii. Blocked ---- Ready
A process is in the blocked state is moved to the ready state when the event for which it has been waiting occurs
viii. Ready ---- Exit
A parent may terminate a child process at any time if a parent associate with that.
ix. Blocked ---- Exit

Creation and termination of process

When a new process is to be added to those concurrently being managed the OS builds the data structures that are used to manage the process and allocates address space to the processor

Reasons for Process Creation

i. New batch job
The OS is provided with a batch job control stream usually on tape or disk. When the OS is prepared to take on a new work it will load the next job sequence of job control command
ii. Interactive log on
A user at a terminal logs on to the system
iii. Created by OS provider service
The OS can create a process to perform functions on behalf of a user program without the user having to wait
iv. Sprung by existing process

For purpose of modularity or to exploit parallelism a user program can declare creation of a number of processes

Reasons for Process Termination

i. Normal completion

The process executes an OS service call to indicate that it has completed running

ii. Time limit exceeded

The process has run longer than the specified total time limit

iii. Memory unavailable

The process requires more memory than the system can provide

iv. Bound variation

The process tries to access memory location that it is not allowed to access

v. Protection error

The process attempts to use a resource or a file that is not allowed to use or it tries to use it in an improper version such as writing to read only file

vi. Arithmetic error

The process tries to prohibit computation e.g. division by zero or tries to state number larger than the hardware can accommodate

vii. Time overrun

The process has waited longer than a specified maximum time for a certain event to occur

viii. I/O failure

An error occurs during I/O such as inability to find a file. Failure to read or write or write after a specified number of times

ix. Invalid instruction

The process attempts to execute a non-existing instruction

x. Data misuse

A piece of data is of the wrong type or is not initialized

xi. Operator / OS intervention

For some reasons the operator or OS has terminated the process e.g. if a deadlock existed

Criteria For Performance Evaluation

i. *Utilization: T*he fraction of time a device is in use. (ratio of in-use time / total Observation time)

ii. *Throughput*: The number of job completions in a period of time. (jobs / second)

iii. Service time the time required by a device to handle a request. (Seconds)

iv. *Queuing time*: Time on a queue waiting for service from the device. (Seconds)

v. *Residence time*: The time spent by a request at a device. vi. Residence time = service time + queuing time.

vii. *Response time*: Time used by a system to respond to a user job. (Seconds)

viii. Think time: The time spent by the user of an interactive system to figure out the next request. (Seconds)

ix. The goal is to optimize both the average and the amount of variation. (But beware the ogre

Predictability)

SUMMARY

The most fundamental concept in a modern OS is the process. The principal function of OS is to create, manage and terminate a process. While processes are active, the OS must see that each is allocated time for execution by the processor, coordinate their activities, manage conflicting demands, and allocate system resources to processes.

To perform its process management functions, the OS maintains a description of each process, or process image, which includes the address space within which the process executes, and a process control block. The latter contains all the information that is required by the OS to manage the process, including its currents state, resources allocated to it, priority and other relevant data.

During its lifetime, a process moves among several states. The most important of these are Ready, Running, and Blocked. A ready process is one that is not currently executing but that is ready to be executed as soon as the OS dispatches it. The running process is that process that is currently being executed by the processor. In multi-processor system, more than one process can be in is state. A blocked process is waiting for the completion of some event, such as an I/O operation.

A running process is interrupted either by an interrupt, which is an event that occurs outside the process and that is recognized by the processor, or by executing a supervisor to call to the OS. In either case, the processor performs a mode switch, transferring control to an operating system routine. The OS, after it has completed necessary work, may resume the interrupted process or switch to some other process.

Additional Reading

https://www.linuxfoundation.org/projects/case-studies/linux/

http://crdd.osdd.net/raghava/slides/prephd/doslec.htm

Review Questions

1. What is an instruction trace?
2. What common events lead to creation of a process?
3. What does it mean to preempt a process?
4. What is swapping and what is its purpose?
5. List four characteristics of a suspended process?
6. For what types of entities does the OS maintain tables of information for management purposes?

7. List three general categories of information in a process control block.
8. Why are two modes (user and kernel)needed?
9. What are the steps performed by an OS to create a new process?
10. What is the difference between an interrupt and a trap?
11. Give three examples of an interrupt.
12. What is the difference between a mode switch and a process switch?

Problems

3.1 The following state transition table is a simplified model of process management, with the labels representing transitions between states of READY, RUN, BLOCKED, and NONRESIDENT.

	READY	RUN	BLOCKED	NONRESIDENT
READY	–	1	–	5
RUN	2	–	3	–
BLOCKED	4	–	–	6

Give an example of an event that can cause each of the above transitions. Draw a diagram if that helps.

3.2 Assume that at time 5 no system resources are being used except for the processor and memory. Now consider the following events:

At time 5: P1 executes a command to read from disk unit 3.
At time 15: P5's time slice expires.
At time 18: P7 executes a command to write to disk unit 3.
At time 20: P3 executes a command to read from disk unit 2.
At time 24: P5 executes a command to write to disk unit 3.
At time 28: P5 is swapped out.
At time 33: An interrupt occurs from disk unit 2: P3's read is complete.
At time 36: An interrupt occurs from disk unit 3: P1's read is complete.
At time 38: P8 terminates.
At time 40: An interrupt occurs from disk unit 3: P5's write is complete.
At time 44: P5 is swapped back in.
At time 48: An interrupt occurs from disk unit 3: P7's write is complete.

For each time 22, 37, and 47, identify which state each process is in. If a process is blocked, further identify the event on which is it blocked.

3.3 Figure 3.9b contains seven states. In principle, one could draw a transition between any two states, for a total of 42 different transitions.

a. List all of the possible transitions and give an example of what could cause each transition.

b. List all of the impossible transitions and explain why.

3.4 For the seven-state process model of Figure 3.9b, draw a queueing diagram similar to that of Figure 3.8b.