2

Processes

2.1 WHAT IS A PROCESS?

Introduction

Earlier the computers could excecute only one program at a time. This program had complete control of the system and had access to all the system. But now, several programs can be loaded into the memory and can be excecuted concurrently.

It leads to compartmentalization of the various programs and these needs resulted in the notion of a process, which is program in execution. A process is the unit of work in a sharing system.

A system consists of collection of operating system processes, executing system code and user processes executing user code. All these processes can execute concurrently, with the CPU multiplexed among them.

It increases the productivity of the computer.

The Process

In general, we say that a process is the program in execution. A process is more than a program code which is sometimes known as text section. In computing, a process is an instance of a computer program that is being executed. It contains the program code and its current activity. Depending on the operating system, a process may be made up of multiple threads of execution that execute instructing concurrently.

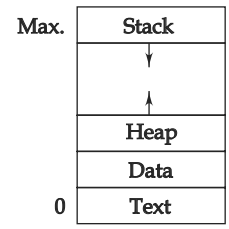

Fig. 2.1 Process in memory

A process generally also includes the process stack, which contains temporary data (such as function parameters, retains addresses and local variables) and a data section

is assigned to it. Process control block is used to store the informmation about the processes and it is also called the data structure, which stores the information about the process. The information of the process is used by the CPU at run time.

The information stored in the PCB are the following:

1. **Process State:** The state may be new, ready, running, waiting, halted and so on.
2. **Program Counter:** The counter indicates the address of the next instruction to be excecuted for that process.
3. **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.
4. **CPU Scheduling Information:** This information includes a process priority, pointers to scheduling queues and any other scheduling parameters.
5. **Memory-management Information:** This information may include such infomation as the value of the base and limit registers the page tables, or the segment tables, depending on the memory system used by the operating system.
6. **Accounting Information:** This information includes the amount of CPU and real-time used, time limits, account numbers, job or process numbers and so on.
7. **I/O Status Information:** This information includes the list of I/O devices allocated to the process, a list to open files and so on.

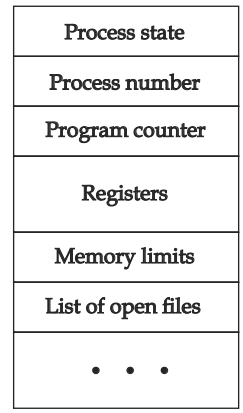

Fig. 2.3 Process control block (PCB)

2.4 THREADS

A thread is a flow of execution through the process code, with its own program counter, system registers and stack. Threads are a popular way to improve application performance through parallelism. A thread is sometimes called a light weight process.

Threads represent a software approach to improve performance of an operating system by reducing the overhead, thread is equivalent to a classical approach. Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control.

Threads have been successufully used in implementing network servers. They also provide a suitable foundation for parallel execution of applications on shared memory

Comparison between Process and Thread

Table 2.1 Process vs Thread

No.	Process	Thread
1.	Process is a program in execution. It is heavy weight or resource intensive.	Threads are flow of execution through process code with its own program counter, system registers are stack. Thread is light weight taking lesser resources than a process.
2.	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3.	In multiple processing environment each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4.	If one process is blocked then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, second thread in the same task can run.
5.	In multiprocessing each process operates independently of the others.	One thread can read, write or change another thread's data.

2.4.3 P-Threads

The P-thread or POSIX thread library is a set of functions that enable multiple threads of execution to do multiple tasks simuntaneousy. It is developed by the IEEE committes in charge of specifying a portable.

- It is fairly low level → you create the thread, you tell it what to do, you wait for it to finish.
- "Communication" is thorough shared memory. A P-threads program cannot span multiple machines.
- No explicit further binding is specified, but you can write a "driven" program in C which makes calls to your further routines.

2.5 SCHEDULERS

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run.

A process comes across various scheduling queues in its life-time. The OS must select the process from these scheduling queues and this selection process is carried out by some appropriate scheduler.

Schedulers are of 3 types:

1. Long-term scheduler
2. Short-term scheduler
3. Medium-term scheduler

2.5.1 Long-term Schedulers

- These are also called job schedulers. Long-term scheduler determines which programs are admitted to the system for processing. It selects processes from pool of processes and loads them into memory for execution.
- The primary distinction between these schedules lies in the frequency of execution. Long-term schedulers excecute much less frequently. Minutes may separate creation of one new process and the next.
- They control the degree of multiprogramming (the number of processes in memory). If degree of multiprogramming is stable, then average rate of process creation must be equal to average departure rate of processes leaving the system. Long term schedulers can afford to take more time to decide which process should be selected for execution.
- The primary objective of a job scheduler is to provide a balanced mix of jobs related to I/O bound processes and CPU processes. If all the processes are I/O bound, the ready queue will almost always be empty. If all processes are CPU bound, I/O waiting queue will almost always be empty. Thus, system will give best performance when there is a proper balance between them.
- On some systems (like UNIX and Microsoft Windows), the long-term scheduler may not be available or minimal. Time-sharing OS has no long-term schedulers. When process changes the state from new to ready, then there is use of long-term schedulers.

2.5.2 Short-term Scheduler

- It is also called a CPU scheduler. It selects the processes that are ready to be executed and it allocates the CPU to one of the selected processes.
- Short-term schedulers have to select new processes for CPU more frequently. It is estimated that, short-term schedules executes atleast once every 100 mil/sec. Because of short life of processes, it has to execute many times hence it should be fast.
- If all processes are I/O bound, ready queue will almost always be empty, and short-term scheduler will have little to do.

2.5.3 Medium-term Schedulers

- Medium-term schedulers are part of the swapping. It removes the process from the memory. It reduces the degree of multiprogramming.
- Medium-term scheduler is in-charge of handling the swapped-out processes.

The diagram for medium-term scheduler is shown in Fig. 2.6:

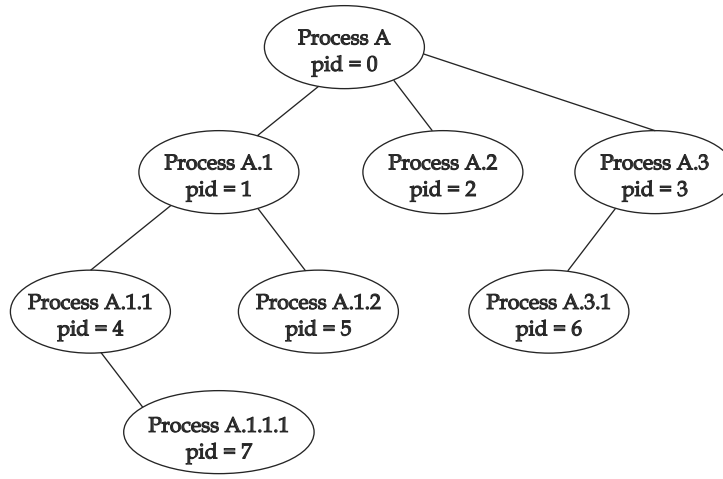

Fig. 2.7 Process creation

- In general, a process may need some resource like CPU time, memory, I/O devices, etc., to accomplish its task. When a process creates a sub-process, it may be able to obtain its resources directly from OS or it may require some other sup-process to get executed and hence provide it the resources.
- As parent process may have to distribute its resources to different child processes, this restricts the process to overload the system by creating many sub-processes.

2.6.2 Process Termination

- A process gets terminated, when it finishes execution of its last statement and asks OS to delete it using 'exit ()' system call.
- When a process finishes its execution it returns a status value (an integer value) to its parent process (by 'wait ()' system call).
- All the resources of the processes like physical and virtual memory, open files, I/O buffers, etc., are deallocated by OS.
- **The termination of process may occur due to the following circumstances.**
  (i) Process finishes its execution
  (ii) One process can cause termination of another process by using the required system call. Such a system call can be involved only by parent of the process that needs to be terminated.
  (iii) Users could arbitrarily kill the jobs.
- **Parent process can terminate the execution of the process due to the following reasons**
  (i) Task assigned to the child is no longer required.
  (ii) The parent process is executing. In such a situation OS will not allow child

process to remain executing.

(iii) Child process has exceeded its usage of some of the resources that it has been allocated.

2.7 INTERPROCESS COMMUNICATION (IPC)

What is IPC?

Interprocess Communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an OS. This allows a program to handle many user requests at the same time. Since even a single user request may result in multiple process running in an OS on user's behalf, the processes need to communicate with each other.

The IPC interface makes this possible. Commonly IPC methods used are shared memory & message passing. Message passing system is more useful and widely used. These mechanisms allow processes to share data and information.

Processes executing in OS can be either independent processes or cooperating processes.

Independent processes: The process that can't affect or be affected by other processes executing in the system. These processes don't share data with other processes.

Cooperating processes: The processes that can affect or be affected by other processes executing in the system. These processes can share data with other processes.

So, any process that shares data (communicate with other processes are cooperating processes. The reasons for providing process cooperation are:

(i) **Information sharing:** Many users may be interested in similar type of information (like shared files), so it is important to allow, to access such information.

(ii) **Computation speed-up:** One way to speed-up execution process is breaking a task into sub-tasks. Each of these sub-tasks can be executed in parallel and we can obtain the desired output earlier.

(iii) **Modularity:** We are able to design modular systrem, i.e., dividing system functions into separate process or threads.

(iv) **Convenience:** We are able to provide the user a flexibility to use multiple applications simultaneously like editing, printing, etc.

Cooperating processes require interprocess communication (IPC) mechanism that allows them to share data and information

There are mainly 2 models of IPC:

1. Shared memory
2. Message passing

2.7.1 Shared Memory System

- IPU using shared memory requires communicating process to establish a region of shared memory.

These features are implemented in the following manner:

(1) Naming

- Process, to communicate must have a way to refer to each other done either by direct or indirect commnication.

(i) Direct communication:

- Each process that wants to communicate must explicitly name the recipient or sender of the communication.

  For this purpose send() and receive() functions are defined as:

  ⇒ send (P, message) – send message to process P

  ⇒ receive (Q, message) – receive message from Q
- Communication link established has the following properties:
  - A link is established automatically between every pair of processes that want to communicate.
  - A link is associated with exactly two processes.
  - Between each pair of processes, there is excactly one link.
- **Disadvantage**

  The scheme's (shown above for sending & receiving disadvantage is the limited modularity of resulting process definitions. Changing the identifier of a process may necessetate examining all other process definitions.

All references to the old identifier must be found so that they can be modified to the new identifier.

(ii) Indirect communication

- In this method of communication, messages are sent and received from *mailboxes* or *ports*.
- A mailbox can be viewed as an object into which messages can be placed by processes and also from which messages can be removed. Each mailbox has a unique identification.
- Two processes can communicate only if the processes have a shared mailbox. The functions used are:

  ⇒ send (A, message) – send message to mailbox A

  ⇒ received (A, message) – reciever message from mailbox
- Communication link established has the following properties:
  - A link is established between a pair of processes only is both members of the pair have a shared mailbox.
  - A link may be associated with more than 2 processes.
  - Between each pair of communicating processes, there may be a number of different links, with each link corresponding to one mailbox.

(2) Synchronization

- Communication beetween processes takes place through calling send() and receive() functions
- Message passing may be either blocking or non-blocking also known as synchronous and asynchronous
  - **Blocking send:** The sending process is blocked until message is received by the mailbox or receiving process.

- **Non-blocking send:** The sending process sends message and resumes operation.
- **Blocking receiver:** The receiver blocks until a message is received.
- **Non-blocking receive:** The receiver retrieves either a valid message or a null.

(3) Buffering

- When communication is direct or indirect, messages exchanged by communicating processes reside in temporary queues.

  These queues can be implemented in 3 ways:

(i) Zero Capacity

- Queue has maximum length of zero
- Thus, link can't have message waiting
- In this, the sender must block until the recipient receives the message.

(ii) Bounded Capacity

- Queue has a finite length of 'n'.
- Thus, at most 'n' messages can reside in it.
- If queue is not full when new message is sent, message is placed in queue and the sender can continue excution without waiting.

(iii) Unbounded Capacity

- Queue length is potentially infinite
- Thus, any number of messages can wait in it
- The sender never blocks.

Summary

- A fundamental function of a sparating system is the execution of a program and an executing program is known as process.
- Process is a dynamic entity, that is, a program in execution. A process is a sequence of instruction executions.
- Transition means change of one state of a process to the other state. A process state transition is a dynamic change in the state of process.
- A process has seven states: new, ready, running, blocked, blocked-suspended, ready-suspended and terminated.
- The process changes its state when there is an event causing a state transition.
- The state transitions are admit (new to ready), dispatch (ready to running), I/O wait (running to blocked), suspended (blocked to blocked-suspended), I/O complete (blocked to ready or blocked-suspended to ready-suspended), activate (ready-suspended to ready), and exit (running to terminated).
- When a job arrives in the ready queue, it becomes a process.
- When a process in the ready queue is selected for execution, it is known as process scheduling.
- The perocess of saving the status of an interrupted process and loading the status of the scheduled process is known as context switching.
- Context switching is a pure overhead from viewpoint of processor time because during context switching no execution is done.

- When a running process is interrupted and the OS assigns another process to the running process and transfers control to it, it is known as process switching.
- A scheduler is a component of the OS that performs the job of scheduling at various levels.

Multiple-Choice Questions

1. A Process Control Block (PCB) does not contain which of the following:
   (a) Code
   (b) Strack
   (c) Heap
   (d) Data
   (e) Program Counter
   (f) Process State
   (g) I/O status information
   (h) bootstrap program
2. The number of processes completed per unit time is known as
   (a) Output
   (b) Throughput
   (c) Efficiency
   (d) Capacity
3. The state of a process is defined by
   (a) the final activity of the process
   (b) the activity just executed by the process
   (c) the activity next be executed by the process
   (d) the current activity of the process
4. Which of the following is not the state of a process?
   (a) New
   (b) Old
   (c) Waiting
   (d) Running
   (e) Ready
   (f) Terminated
5. The process control block is
   (a) Process type variable
   (b) Data structure
   (c) A secondary storage section
   (d) A Block in memory
6. The entry of all the PCBs of the current processes is in
   (a) Process register
   (b) Program counter
   (c) Process table

(b) Communication between two process
(c) Communication between two threads of same process
(d) None of the above

16. A set of processes is deadlock if
(a) each process is blocked and will remain so forever
(b) each process is terminated
(c) all processes are trying to kill each other
(d) none of the above

17. A process stack does not contain
(a) function parameters
(b) local variables
(c) return addresses
(d) PID of child process

18. Which system call returns the process identifier of a terminated child?
(a) Wait
(b) exit
(c) fork
(d) get

19. The address of the next instruction to be executed by the current process is provided by the
(a) CPU registers
(b) program counter
(c) proces stack
(d) pipe

20. Which of the following need not necessarily be saved on a context switch between proceses?
(a) General purpose registers
(b) Translation look-aside buffer
(c) Program counter
(d) All of these

Answers

1. (h)	2. (b)	3. (d)	4. (b)	5. (b)	6. (c)
7. (d)	8. (a)	9. (a) and (d)		10. (a)	11. (d)
12. (a)	13. (d)	14. (a)	15. (b)	16. (a)	17. (d)
18. (a)	19. (b)	20. (b)			