OS

5 Marks

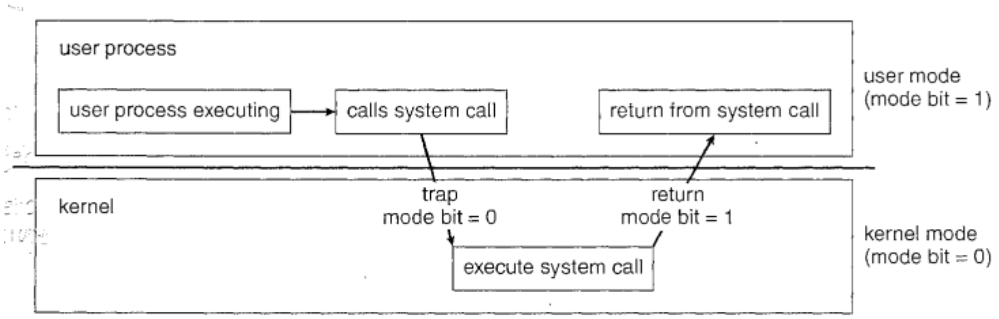

Figure 1.10 Transition from user to kernel mode.

Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt

Operating-System Debugging

- Debugging is finding and fixing errors, or bugs
- OS generate log files containing error information
- Failure of an application can generate core dump file capturing memory of the process
- Operating system failure can generate crash dump file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
    - Sometimes using *trace listings* of activities, recorded for analysis
    - Profiling is periodic sampling of instruction pointer to look for statistical trends

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
- Context of a process represented in the PCB
- Context-switch time is overhead;
- Time depends on hardware support
- Some hardware provides multiple sets of registers per CPU
- multiple contexts loaded at once

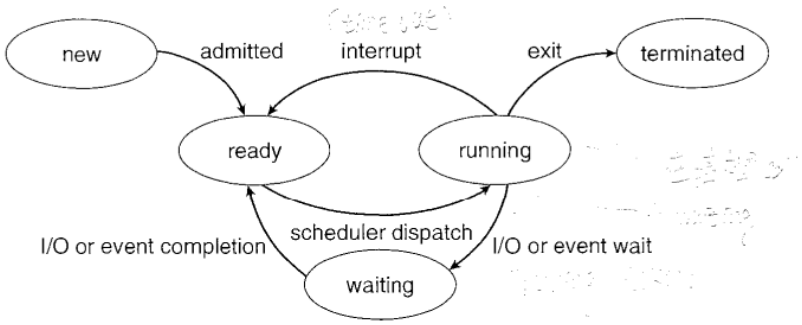

Figure 3.2 Diagram of process state.

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
    - switching context
    - switching to user mode
    - jumping to the proper location in the user program to restart that program
- Dispatch latency – time it takes for the dispatcher to stop one process and start another running

10 Marks

Q(2) Purpose of System call + Figure (2.7)

- *Purpose of system calls are given as follows:*
  - *Basic Purpose: Calls provide basic functionality to users to operate the operating system.*
  - *Process Control: Systems call loads, execute and create processes and terminate when the user's task is finished with the process.*
  - *File Management: It provides file management such as creating a file, deleting it, open, close, and saving it. It also provides read, write and reposition functionalities.*
  - *Device Management: All hard disks are managed by system calls such as requesting for a device, releasing the device, reading and writing the device.*
  - *Information Maintenance: System calls help in making information maintenance such as get/set time or date, get/set data of system, processes, files or attributes of device.*
  - *Communication between processes: Systems calls are use for Communication purpose as they help in creating and deleting communications, sending or receiving messages. They help in attaching or detaching remote devices and in transfer or status information.*

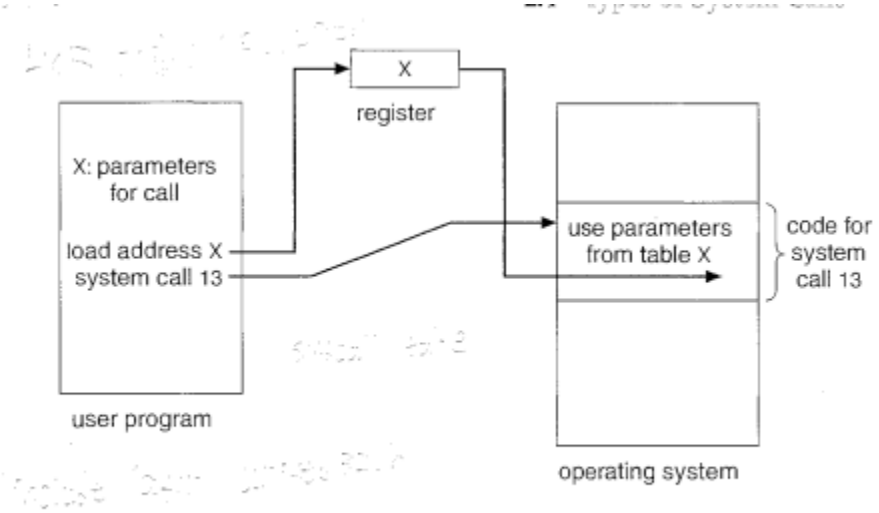

Figure 2.7 Passing of parameters as a table.

Process Concept + Figure(3.1)(ch3)

- An operating system executes a variety of programs:
    - Batch system – jobs/processes
    - Time-shared systems – user programs or tasks
- Process – a program in execution; process execution must progress in sequential fashion
- Multiple parts
    - The program code, also called text section
    - Current activity including program counter, processor registers
    - Stack containing temporary data
        - Function parameters, return addresses, local variables
    - Data section containing global variables
    - Heap containing memory dynamically allocated during runtime
- Program is a passive entity stored on disk (executable file).
- Process is an active entity with a program counter specifying the next instruction to execute and a set of associated resources.
    - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
    - Consider multiple users executing the same program

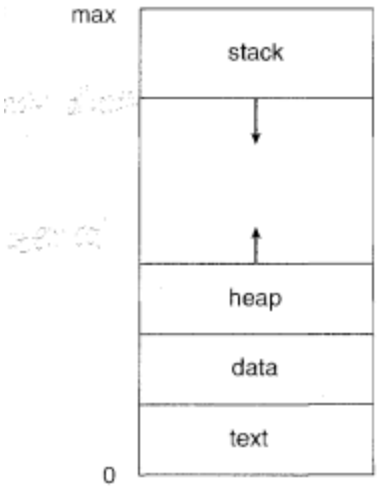

Figure 3.1 Process in memory.

Multilevel queue + Multilevel feedback queue (ch5)

Multilevel queue

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes;
- A process can move between the various queues; aging can be implemented this way

Multilevel feedback queue

- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

CPU Scheduler + Scheduling Criteria(Ch5)

CPU Scheduler

- Short-term scheduler selects from among the processes in the ready queue, and allocates the CPU to one of them.
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state

2. Switches from running to ready state
3. Switches from waiting to ready
4. Terminates

- Scheduling under 1 and 4 is nonpreemptive
- All other scheduling is preemptive
    - Consider access to shared data
    - Consider preemption while in kernel mode
    - Consider interrupts occurring during crucial OS activities

Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Thread Cancellation (Ch4)

- Terminating a thread before it has finished
- Having made the cancellation request, *pthread_cancel ( )* returns immediately; that is it does not wait for the target thread to terminate.
- Thread attributes that indicate cancellation state and type
- Two general states
    - Thread can not be canceled.
        - PTHREAD_CANCEL_DISABLE
    - Thread can be canceled
        - PTHREAD_CANCEL_ENABLE
        - Default
- When thread is cancelable, there are two general types
    - Asynchronous cancellation terminates the target thread immediately
        - PTHREAD_CANCEL_ASYNCHRONOUS
    - Deferred cancellation allows the target thread to periodically check if it should be canceled
        - PTHREAD_CANCEL_DEFERRED
        - Cancel when thread reaches 'cancellation point'

Thread Overview + Multithreaded programming Benefits + Figure(4.1)

Thread Overview

- Threads are mechanisms that permit an application to perform multiple tasks concurrently.
- Thread is a basic unit of CPU utilization
    - Thread ID
    - Program counter
    - Register set
    - Stack
- A single program can contain multiple threads
    - Threads share with other threads belonging to the same process its code section, data section, and other operating-system resources.

Benefits

- **Responsiveness**

  Interactive application can delegate background functions to a thread and keep running

- **Resource Sharing**

  Several different threads can access the same address space

- **Economy**

  Allocating memory and new processes is costly. Threads are much 'cheaper' to initiate.

- **Scalability**

  Use threads to take advantage of multiprocessor architecture

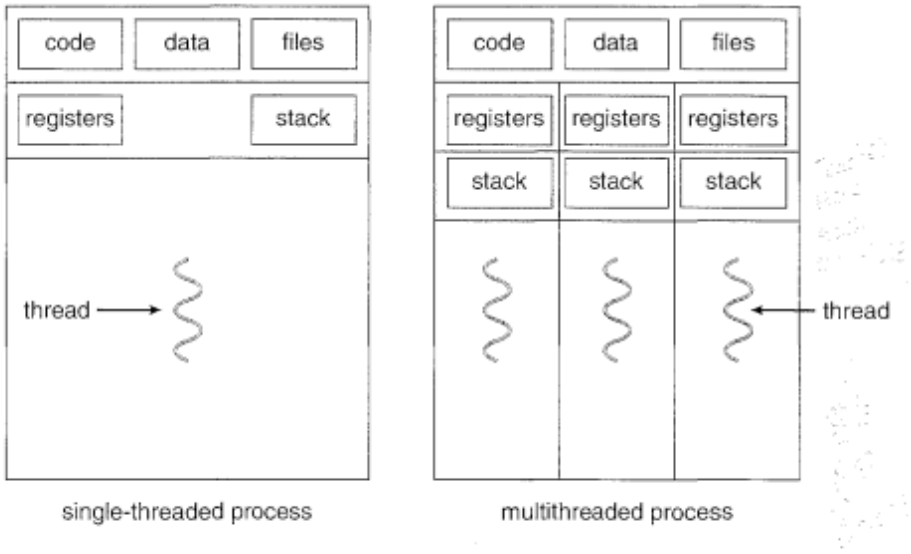

Figure 4.1 Single-threaded and multithreaded processes.

Process Creating & Process Termination + Figure(3.11)

Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via a process identifier (pid)
- Resource sharing options
    - Parent and children share all resources
    - Children share subset of parent's resources
- Execution options
    - Parent and children execute concurrently
    - Parent waits until children terminate
- Address space
    - Child duplicate of parent
    - Child has a program loaded into it
- UNIX examples
    - fork() system call creates new process
    - exec() system call used after a fork() to replace the process' memory space with a new program

Process Termination

- Process executes the last statement and then asks the operating system to delete it using the exit() system call.
    - Returns status data from child to parent (via wait())
    - Process' resources are deallocated by operating system
- Parents may terminate the execution of children processes using the abort() system call.
- If a process terminates, then all its children must also be terminated.
    - cascading termination. All children, grandchildren, etc. are terminated.
    - The termination is initiated by the operating system.

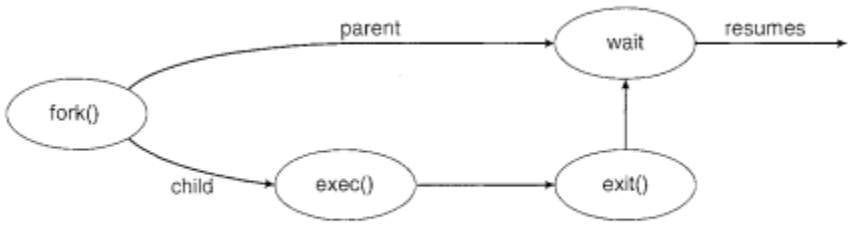

Figure 3.11 Process creation using `fork()` system call.

Q(3.4) two models of interprocess communications + Figure (3.13)

- The two models of interprocess communication are the message-passing model and the shared-memory model.
- Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided. It is also easier to implement than shared memory for intercomputer communication.
- Shared memory allows maximum speed and convenience of communication, since it can be done at memory transfer speeds when it takes place within a computer. However, this method compromises on protection and synchronization between the processes sharing memory.

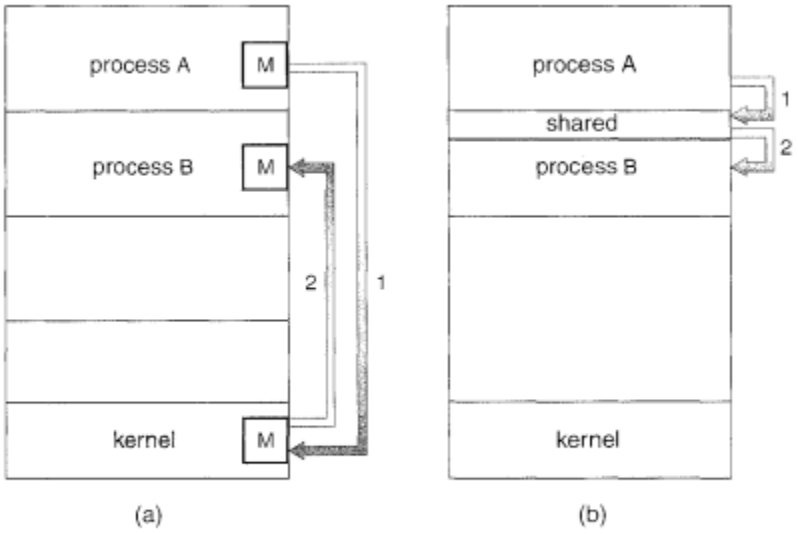

Figure 3.13 Communications models. (a) Message passing. (b) Shared memory.