CSE 4501 Operating Systems
Chapter 3: Processes

Practice Exercises *(Will help to understand questions those may come in the quiz, mid and final examinations)*

Question-1. Palm OS provides no means of concurrent processing. Discuss three major complications that concurrent processing adds to an operating system.

a. A method of time sharing must be implemented to allow each of several processes to have access to the system. This method involves the preemption of processes that do not voluntarily give up CPU (by using a system call, for instance) and the kernel being reentrant (so more than one process may be executing kernel code concurrently.
b. Processes and system resources must have protections and must be protected from each other. Any given process must be limited in the amount of memory it can use and the operations it can perform on devices like disks.
c. Care must be taken in the kernel to prevent deadlocks between processes, so processes aren't waiting for each other's allocated resources

Question-2. The Sun UltraSPARC processor has multiple register sets. Describe what happens when a context switch occurs if the new context is already loaded into one of the register sets. What happens if the new context is in memory rather than in a register set and all the register sets are in use?

The CPU current-register-set pointer is changed to point to the set containing the new context, which takes very little time. If the context is in memory, one of the contexts in a register set must be chosen and be moved to memory, and the new context must be loaded from memory into the set. This process takes a little more time than on systems with one set of registers, depending on how a replacement victim is selected.

Question-3. When a process creates a new process using the fork() operation, what are shared between the parent process and the child process?

Only the shared memory segments are shared between the parent process and the newly forked child process. Copies of the stack and the heap are made for the newly created process.

Question-4. Describe the differences among short-term, medium-term, and long-term scheduling.

- Short-term scheduling is also called as CPU Scheduling. It selects among the processes that are ready to execute and allocates the CPU to one of them. It selects a new process for the CPU frequently. It executes at least once in every 100 milliseconds. Hence it is fast
- Medium-term scheduling is having advantage to remove processes from memory and thus reduce the degree of multiprogramming. The process can be reintroduced into the memory, and its execution can be continued where it left off. This method is called swapping. This is done by medium-term scheduler.
- Long -term scheduling is also called as job scheduling. It selects processes from this pool and loads them into memory for execution. Long-term scheduler executes much less frequently. It also controls degree of multi programming. It select a good process mix of I/O bound and CPU-bound process.

Question-5. What is context-switch? Describe the actions taken by a kernel to context-switch between processes.

Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a **context switch**

The process of context switching between the processes is accomplished by the kernel. This process is performed as follows:

a. In reply to the clock interrupt, Operating System stores the value of the Program Counter and the user Stack Pointer of the presently implementing process, and handovers charge to the kernel clock interrupt handler.

b. The clock interrupt handler hold back the remaining registers, along with other machine status such the status of the floating point registers in the Process Control Block of the process.
c. The Operating System call upon the scheduler to decide the next process that has to be implemented.
d. The Operating System takes the status of the next process from its Process Control Block and fix up the registers. This restore task takes the processor back to the state in which the process was earlier interrupted, implementing in user code with user mode privileges.

Question-6. What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.

- **Synchronous and asynchronous communication**
- **Automatic and explicit buffering**
- **Fixed-sized and variable-sized messages**

a. **Synchronous and asynchronous communication:**

- A benefit of symmetric communication is that it allows a rendezvous between the sender and receiver. At the programmer level, neither process has to block its execution which can result in better performance.
- The disadvantage of asymmetric communication is that, blocking send is a rendezvous and may not be required and the message could be delivered asynchronously and received at a point of no interest to the sender. As a result, message-passing systems often provide both forms of synchronization.
- And it is more difficult to program since the programmer must guarantee that the message arrive at the receiver when it is needed. At the system level, asymmetric is more complicated since it requires kernel-level buffering.

b. **Automatic and explicit buffering:**

- Automatic buffering provides a queue with indefinite length. Thus ensuring that the sender will never have to block while waiting to copy a message. There are no specifications how automatic buffering will be provided. One schema may reserve sufficiently large memory where much of the memory is wasted.

- Explicit buffering specifies how large the bugger is. In this situation, the sender may be blocked while waiting for available space in the queue. However, it is less likely that memory will be wasted in explicit buffering.

c. Fixed-sized and variable sized messages:

- The implications of this are mostly related to buffering issues with fixed-size messages (a buffer with a specific size can hold a known number of messages). Fixed-Sized messages are easier to implement at the kernel-level but require slightly more effort on the part of the programmer.
- Variable sized messages are somewhat more complex for the kernel but somewhat easier for the programmer. The number of variable-sized messages that can be held by such a buffer is unknown.

Question-7. Write details about followings

a. Shared Memory

b. Message passing

(Details given in book and slides)

Question-8. What are the benefits and disadvantages of shared memory and message passing?

Shared Memory

Advantages:

- Memory communication is faster
- Kernel is not involved
- Any number process can communicate same time

Disadvantages:

- All the processes that use the shared memory model need to make sure that they are not writing to the same memory location.
- Shared memory model may create problems such as synchronization and memory protection.
- Complex implementation
- Separate system calls are required to handle shared memory

Message Passing

Advantages:

- Hardware can be simpler
- Communication explicit
- Synchronization is naturally associated with sending messages, reducing the possibility for errors introduced by incorrect synchronization
- Easier to use sender-initiated communication
- It is easier to build parallel hardware using message passing model as it is quite tolerant of higher communication latencies.

Disadvantages:

- High message passing overhead
- Complex to program
- Can dramatically slow down the system if huge amount of data needs to be sent.

Question-9. Definitions:

a. Zombie process

A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process.

b. Orphan process

A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.

c. Independent process

Independent process is the process that can not affect or be affected by the other processes. Independent processes does not share any data like temporary or persistent with any other process

d. Cooperating process

Cooperating process is affect or be affected by the other processes executing in the system. Cooperating process shares data with other processes.

e. Direct communication

In direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication.

f. Indirect communication

In indirect communication, the messages are sent to and received from mailboxes, or ports.

g. Blocking send

The sending process is blocked until the message is received by the receiving process or by the mailbox.

h. Blocking receive

The receiver blocks until a message is available.

i. Non-blocking send

The sending process sends the message and resumes operation.

j. Non-blocking receive

The receiver retrieves either a valid message or a null.

k. Zero capacity Buffering

The queue has a maximum length of zero; thus, the link cannot have any messages waiting in it. In this case, the sender must block until the recipient receives the message.

l. Bounded capacity Buffering

The queue has finite length *n;* thus, at most *n* messages can reside in it. If the queue is not full when a new message is sent, the message is placed in the queue (either the message is copied or a pointer to the message is kept), and the sender can continue execution without waiting. The link's capacity is finite, however. If the link is full, the sender must block until space is available in the queue.

m. Unbounded capacity Buffering

The queue's length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks.

Question-10. How processes communicate with each other via direct communications and indirect communications?

In **direct communication**, each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send() and receive() primitives are defined as:

- **send(P, message)—**Send a message to process P.
- **receive(Q, message)—**Receive a message from process Q.

This scheme exhibits *symmetry* in addressing; that is, both the sender process and the receiver process must name the other to communicate. A variant of this scheme employs *asymmetry* in addressing. Here, only the sender names the recipient; the recipient is not required to name the sender. In this scheme, the send() and receive() primitives are defined as follows:

send(P, message): Send a message to process P.

receive(id, message): Receive a message from any process; the variable *id* is set to the name of the process with which communication has taken place.

With **indirect communication**, the messages are sent to and received from **mailboxes**, or **ports**. A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed. Each mailbox has a unique identification. For example, POSIX message queues use an integer value to identify a mailbox. In this scheme, a process can communicate with some other process via a number of different mailboxes. Two processes can communicate only if the processes have a shared mailbox, however. The send() and receive() primitives are defined as follows:

- **send(A, message)—**Send a message to mailbox A.
- **receive(A, message)—**Receive a message from mailbox A.

Question-11. What are the benefits and disadvantages of direct communication and indirect communication?

Direct communication
Advantages:

- Explicit
- Simple

Disadvantages:

- Limited modularity of the resulting process definitions
- Changing an ID of a process -> may need to examine all other process definitions.

Indirect communication
Advantages:

- More flexible
- No question who received the message from a mailbox

Disadvantages:

- Operating system must provide mailbox mechanism
- Mailbox ownership may be passed -> could result in multiple receivers.

Question-12. Write down the properties of direct communication link and indirect communication link.

Properties of direct communication link:

- Links are established automatically
- A link is associated with exactly one pair of communicating processes
- Between each pair there exists exactly one link
- The link may be unidirectional, but is usually bi-directional

Properties of indirect communication link:

- Link established only if processes share a common mailbox
- A link may be associated with many processes
- Each pair of processes may share several communication links
- Link may be unidirectional or bi-directional

Question-13. Write down the advantages of process cooperation.

(also the answer to reasons to provide process cooperation)

- **Information sharing:** Since several users may be interested in the same piece of information (for instance, a shared file), we must provide an environment to allow concurrent access to such information.
- **Computation speedup**: If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others. Notice that such a speedup can be achieved only if the computer has multiple processing elements (such as CPUs or I/O channels).
- **Modularity**. Helps to construct the system in a modular fashion, dividing the system functions into separate processes or threads.
- **Convenience**. Even an individual user may work on many tasks at the same time. For instance, a user may be editing, printing, and compiling in parallel.

Question-14. Describe about resource sharing between parent and child processes. Compare among them.

Resources: CPU time, memory, files, I/O devices
Resource sharing options:

- Parent and children share all resources
- Children share subset of parent's resources: The parent partitions its resources among its children, or it shares some resources (such as memory or files) among several of its children. Restricting a child process to a subset of the parent's resources prevents any process from overloading the system.
- Parent and child share no resources

Question-15. What is Process Control Block (PCB)? Write down the components of a PCB.

A process control block (PCB) is a data structure used by an operating system to manage processes.

Components of PCB:

- Process state – running, waiting, halted etc.
- Program counter – location of instruction to next execute

- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

Question-16. Write a short note on process state with state diagram.

Process State:

As a process executes, it changes **state**. The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:

• **New**. The process is being created.
• **Running**. Instructions are being executed.
• **Waiting**. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
• **Ready**. The process is waiting to be assigned to a processor.
• **Terminated**. The process has finished execution.

These names are arbitrary, and they vary across operating systems. The states that they represent are found on all systems, however. Certain operating systems also delineate process states more finely. It is important to realize that only one process can be *running* on any processor at any instant. Many processes may be *ready* and *waiting*, however.

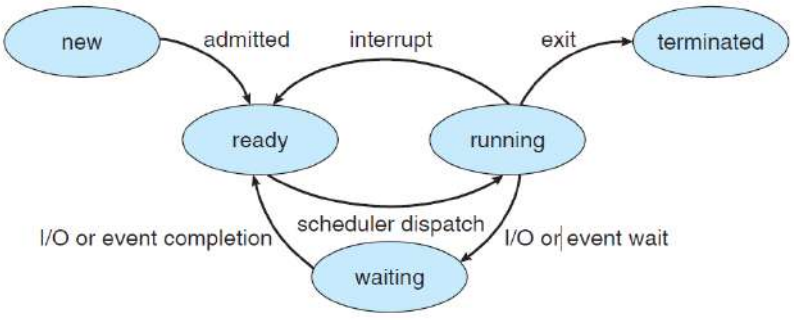

Figure 3.2 Diagram of process state.

Question-17. How a process is stored in the main memory while being executed? Describe with different parts in the memory block.

A process is a program in execution.

A process is more than the program code, which is sometimes known as the **text section**.

It also includes the current activity, as represented by the value of the **program counter** and the contents of the processor's registers.

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

A process may also include a **heap**, which is memory that is dynamically allocated during process run time. The structure of a process In memory is shown below:

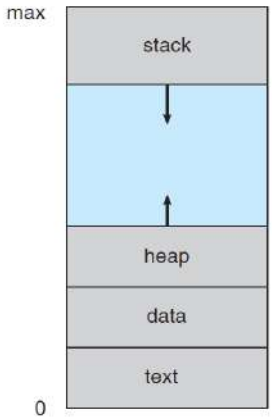

Figure 3.1 Process in memory.

A program is a *passive* entity, such as a file containing a list of instructions stored on disk (often called an **executable file**), whereas a process is an *active* entity, with a program counter specifying the next instruction to execute and a set of associated resources. A program becomes a process when an executable file is loaded into memory.

Question-18. Programming related questions like output after a specific line, number of child processes, PID values, etc. will come in the examination.