HOME ASSIGNMENT-2

BY: - Shailee Mehta (121048)

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. Give an example of an event that can cause each of the above transitions. Draw a diagram if that helps. (FIGURE in pdf)**

   ANSWER:
   [1] **READY to RUN**- If a process is allocated the processor by the dispatcher.
   [2] **RUN to READY**- By a time-quantum expiration.
   [3] **RUN to BLOCKED**- Request for I/O operations.
   [4] **BLOCKED to READY**- Awaited request serviced.
   [5] **BLOCKED to NONRESIDENT/READY to NONRESIDENT**- When the memory is overcommitted, and a process is temporarily swapped out of memory.

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.

   ANSWER:
   At time 22-
   P1: Blocked for I/O
   P3: Blocked for I/O
   P5: Ready Running
   P7: Blocked for I/O
   P8: Ready Running

   At time 37-
   P1: Ready Running

P3: Ready Running
P5: Blocked for I/O
P7: Blocked for I/O
P8: Ready Running

At time 47-
P1: Ready Running
P3: Ready Running
P5: Ready Suspended
P7: Blocked for I/O
P8: Exit

3. **You have executed the following C program:**

```
main (){
int pid;
pid = fork ();
printf (“%d \n”, pid);
}
```

What are the possible outputs, assuming the fork succeeded?

ANSWER:
Any child PID= 0, either ways will be.

4. **List reasons why a mode switch between threads may be cheaper than a mode switch between processes.**

   ANSWER:
   The reasons why a mode switch between threads is better than a process switch is-
   - The processes require a context switch which involves going from user to kernel mode. It utilizes a lot of kernel time which becomes a costly affair.
   - The control blocks for processes hold more state information than threads’, so the amount of information to move during the thread switching is less than for process context switching.

5. **List three advantages of ULTs over KLTs.**

   ANSWER:
   - The process does not switch to the kernel mode to do thread management. This saves the overhead of two mode switches between user to kernel and kernel back to user.
   - Scheduling can be application specific. The scheduling algorithm can be changed to the application without disturbing the OS scheduler.
   - They are portable pieces of codes. No changes are required to the underlying kernel to support ULTs. The threads library is a set of application-level utilities shared by all applications.

6. **List two disadvantages of ULTs compared to KLTs.**

ANSWER:

- When a ULT executes a system call, not only is that thread blocked, but also all of the threads within the process are blocked.
- Here,a multithreaded application cannot take advantage of multiprocessing. A kernel assigns one process to only one processor at a time. Therefore, only a single thread within a process can execute at a time.

7. **In the discussion of ULTs versus KLTs, it was pointed out that a disadvantage of ULTs is that when a ULT executes a system call, not only is that thread blocked, but also all of the threads within the process are blocked. Why is that so?**

   ANSWER:
   YES. Switching between two threads is a job in user context which uses the usual resources. Whereas KLT are kernel context jobs and are rather viewed as independent processes. So, blocking of a KLT wouldn't disturb the rest of the process but since ULTs aren't kernel level they would block, blocking the process with them.

8. **Consider an environment in which there is a one-to-one mapping between user-level threads and kernel-level threads that allows one or more threads within a process to issue blocking system calls while other threads continue to run. Explain why this model can make multithreaded programs run faster than their single-threaded counterparts on a uniprocessor computer.**

   ANSWER:
   In a multithreaded program, one KLT may be blocked, while the other KLTs can continue to run. On uniprocessors, a process that would otherwise have to block for all these calls can continue to run its other threads. The single KLT and the process (in the ULT environment) will however be blocked for too long as I/O process requested for is usually very slow.

9. **If a process exits and there are still threads of that process running, will they continue to run?**

   ANSWER:
   NO, once a process exits all its threads will be killed along with it.

10. **What is the distinction between competing processes and cooperating processes?**
    ANSWER:

Topic	Competing Process	Cooperating Process
Dependence on other processes	Competing process does its work independent of any other process present.	Cooperating process does its work in accordance with the other present processes.

Awareness	Completely unaware, will compete for resources.	Completely aware .Share the resources with some other process and at times even complete a task together with other processes.
Synchronization	There is a careful isolation done among all the processes.	The processes are made to communicate and share with each other.

11. What is the difference between strong and weak semaphores?

ANSWER:

Strong Semaphore	Weak Semaphore
Strict FIFO implementation.	The order in which the process should be removed from the waiting queue is undefined.
Mostly used by all the Operating System	Rarely used by any operating system

12. What is a monitor?

ANSWER:
Monitor is a synchronization construct that allows the threads to wait for some event to occur and assure mutual exclusion between them. It is helpful for multiprogramming. With the help of monitors only one thread will be executed at a time.

13. What is the distinction between *blocking* and *non-blocking* with respect to messages?

ANSWER:
Send blocking: The sending process is blocked until the message is received.
Receive blocking:

[1] If a message has been sent and is available, the message will be received and normally executes ahead.

[2] If there is no message waiting, then either, the process is blocked until a message arrives or the process executes stopping to wait.

14. Is busy waiting always less efficient (in terms of using processor time) than a blocking wait? Explain.

ANSWER:
No, rather for shorter period process, waiting is very much less expensive than block as blocking needs context switch. Thus it will take more processor time.

15. Consider the following definition of semaphores:

```
void semWait(s)
{
if (s.count > 0) {
s.count--;
}
else {
place this process in s.queue;
block;
}
}
void semSignal (s)
{
if (there is at least one process blocked on
semaphore s) {
remove a process P from s.queue;
place process P on ready list;
}
else
s.count++;
}
```

Compare this set of definitions with one given below. Note one difference: With the preceding definition, a semaphore can never take on a negative value. Is there any difference in the effect of the two sets of definitions when used in programs?

That is, could you substitute one set for the other without altering the meaning of the program?

```
struct semaphore {

int count;

queueType queue;

};

void semWait(semaphore s)

{

s.count--;

if (s.count < 0) {

/* place this process in s.queue */;

/* block this process */;

}

}

void semSignal(semaphore s)

{

s.count++;

if (s.count<= 0) {

/* remove a process P from s.queue */;

/* place process P on ready list */;

}

}
```

ANSWER:

Yes they can be substituted without altering meanings as they are the same.