COMP 346: Operating Systems
Theory Assignment 2

Duc Nguyen

Gina Cody School of Computer Science and Software Engineering
Concordia University, Montreal, QC, Canada

Winter 2020

Contents

1 User versus kernel threads models

1.1 Problems:

- What are the main differences between the user-kernel threads models?
- Which one of these models is likely to trash the system if used without any constraints?

1.2 Answer:

- Two main differences are:
  - User-level threads are unknown by the kernel, whereas the kernel is aware of kernel threads
  - Kernel threads need not be associated with a process whereas every user thread belongs to a process

2 Light-weight processes

2.1 Problem:

- Why threads are referred to as light-weight processes?
- What resources are used when a thread is created?
- How do they differ from those used when a process is created?

2.2 Answer:

- Threads are sometimes called lightweight processes because they have their own stack but can access shared data.
- Threads are smaller than processes, so they needs less resources. Threads allocate a small data structure to hold a register set, stack, and priority. A process allocates a PCB, which is a rather large data structure. Context switching with a thread is much quicker than a process.

3 Shared memory problem

3.1 Problem:

- Does shared memory provide faster or slower interactions between user processes?
- Under what conditions is shared memory not suitable at all for inter-process communications?

3.2 Answer:

- In shared memory, the communication between the processes take place through shared address space. So, in shared memory system, multiple processes can share the virtual memory space.

  Thus, shared memory provides fastest interactions between user processes as the memory segment is mapped by the operating system in the address space of several processes.

- he conditions in which the shared memory is not suitable for inter-process communications are as follows:

  - When more CPUs are added corresponding to a shared memory section, it can lead to an increase in traffic on shared memory, which will increase the traffic associated with the cache. So, the shared memory models do not work across multiple machines.
  - In current scenario, due to complexity of the systems as well as the tasks, the requirement of processors increases, so the cost as well as the difficulty to design shared memory machines according to the increasing complexity of processors increases.
  - The shared memory models create synchronization problems and not suitable for producer consumer problems
  - It is also unsuitable for the conditions of bounded buffer and unbounded buffer.

4 Concurent processes

4.1 Problem:

- Consider three concurrent processes A, B, and C, synchronized by three semaphores mutex, goB, and goC, which are initialized to 1, 0 and 0 respectively:

```
Process A
-----------
wait (mutex)
...
signal (goB)
...
signal (mutex)
```

```
Process B
-----------
wait (mutex)
...
wait (goB)
signal (goC)
...
signal (mutex)
```

```
Process C
-----------
wait (mutex)
...
wait (goC)
...
signal (mutex)
```

Does there exist an execution scenario in which:

  - (i) All three processes block permanently?
  - (ii) Precisely two processes block permanently?
  - (iii) No process blocks permanently? Justify your answers.

4.2 Answer:

5 Swapping/relocation system

5.1 Problem:

In a swapping/relocation system, the values assigned to the <base, limit> register pair prevent one user process from writing into the address space of another user process. However, these assignment operations are themselves privileged instructions that can only be executed in kernel models ode. Is it conceivable that some operating-system processes might have the entire main memory as their address space? If this is possible, is it necessarily a bad thing? Explain.

5.2 Answer:

Usually, the operating system also resides in the low memory partition of the main memory. Other user processes use the high memory partition. It is possible that some operating-system process might have the enitre memory as their address space. Following are the implications -

- slows down other processes that are waiting for the main memory.
- New processes are queued in the Secondary Storage. Resulting in Low Disk Space
- Processes do not interact with each other. Interaction might be necessary for inter process communication.

6 P and V as a non-critical section

6.1 Problems:

Generally, both P and V operation must be implemented as a critical section. Are there any cases when any of these two operations can safely be implemented as a non-critical section? If yes, demonstrate through an example when/how this can be done without creating any violations. If no, explain why these operations must always be implemented as critical sections.

6.2 Answer:

These operations can be implemented as non critical sections. For example, there are three process have to perform addition function on any two numbers. But the addition function is present as a function between P() and V() i.e if one process is using it the others cannot, So when a single process is using the addition function all other have to wait as per the P() function. It could have been possible that all the three process uses the addition function at once if it was not placed inside a P() and V() functions. Here the addition function is for local variables which is different for different process but if it had been global variables that would have been same for all the processes, it comes under the case of shared resource then the addition function should be placed between P() and V() function

7 Multiprogramming

7.1 Problem:

What is the potential problem of multiprogramming?

7.2 Answer:

Problems Stealing or copying a user's files; Writing over another program's (belonging to another user or to the OS) area in memory; Using system resources (CPU, disk space) without proper accounting; Causing the printer to mix output by sending data while some other user's file is printing.