Operating Systems

End Semester Revision

Processes and Threads

The Process Model

- A process is an abstraction of a running program
- It enables doing several things at the same time – support the ability to have (pseudo) concurrent operation
- In a multiprogramming system, the CPU switches from process to process quickly, running each for tens or hundreds of milliseconds
- All the runnable software on the computer, sometimes including the operating system, is organized into a number of sequential processes
- To understand the system, it is much easier to think about a collection of processes running in (pseudo) parallel
- This rapid switching back and forth is called multiprogramming

Process Creation

Events which cause processes creation:

1. System initialization.
2. Execution of a process creation system call by a running process.
3. A user request to create a new process.
4. Initiation of a batch job.

In all these cases, a new process is created by having an existing process execute a process creation system call.

Process Termination

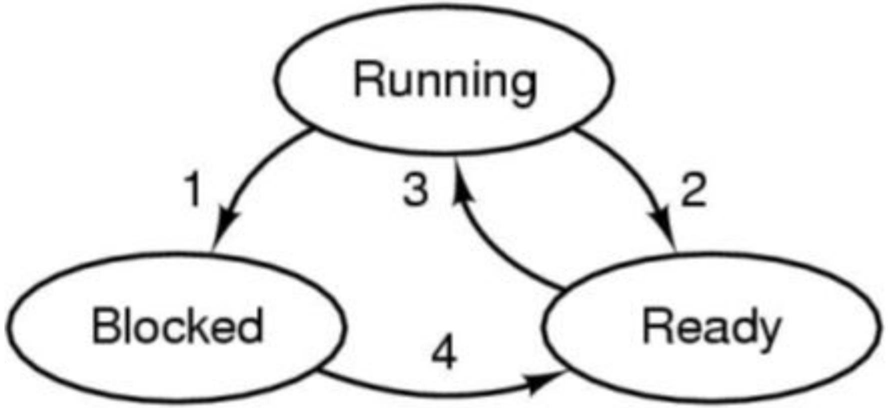
Events which cause process termination:

1. Normal exit (voluntary).
2. Error exit (voluntary).
3. Fatal error (involuntary).
4. Killed by another process (involuntary).

Process States

Three states a process may be in:

1. Running (actually using the CPU at that instant).
2. Ready (runnable; temporarily stopped to let another process run).
3. Blocked (unable to run until some external event happens).



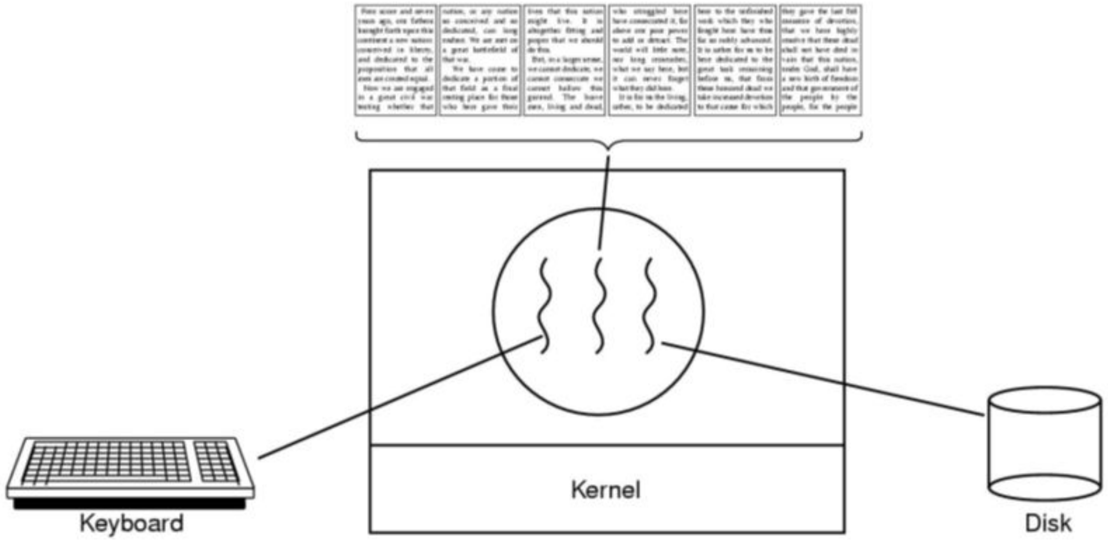
1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Threads

- In many applications, multiple activities are going on at once
- Threads give the ability for the parallel entities to share an address space and all of its data
    - not possible with multiple processes (with their separate address spaces)
- Threads are lighter weight than processes, so they are easier (i.e., faster) to create and destroy
- Threads allow computing and IO activities to overlap

Thread Usage - A word processor

1. First thread interacts with the user.
2. Second thread handles reformatting in the background.
3. Third thread handles the disk backups.



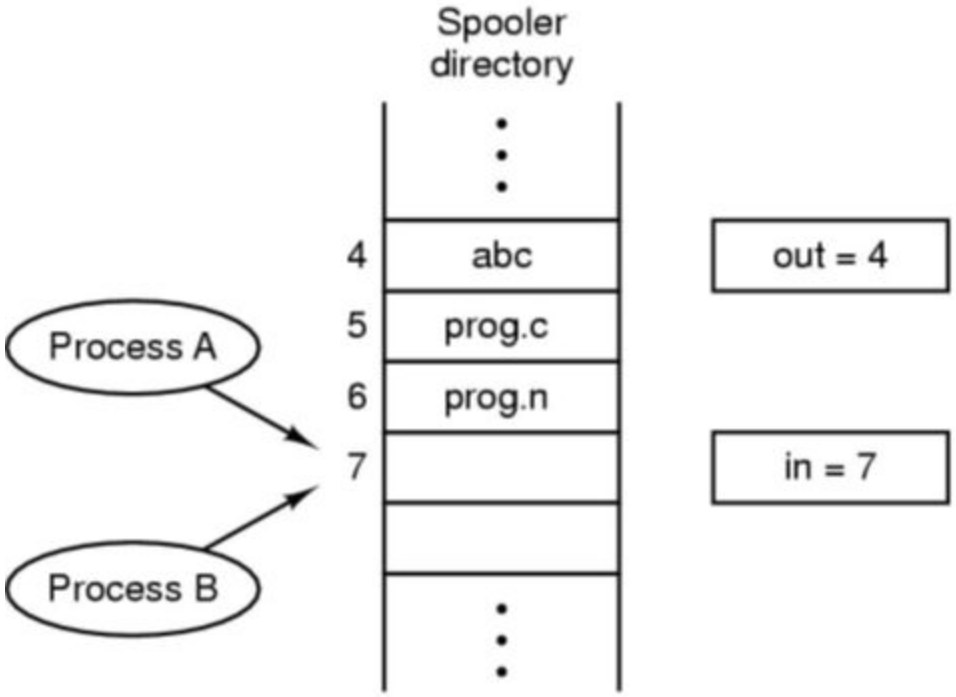
Inter-process Communication (IPC)

Processes frequently need to communicate with other processes

- To pass information from process to another
- To make sure two or more processes do not get in each other’s way
- To enable proper sequencing when dependencies are present

Race Conditions

- Two or more processes are reading or writing some shared data and the final result depends on who runs precisely when
- Ex. two processes want to access the printer spooler directory at the same time



Conditions required to avoid race condition

1. No two processes may be simultaneously inside their critical regions.
   – Mutual Exclusion (mutex)
2. No assumptions may be made about speeds or the number of CPUs.
   – No Assumption
3. No process running outside its critical region may block other processes.
   – Progress
4. No process should have to wait forever to enter its critical region.
   – No Starvation

Mutual Exclusion

- We want to use mutual exclusion to synchronize access to shared resources
    - This allows us to have larger atomic blocks
- Code that uses mutual exclusion to synchronize its execution is called a critical region (or critical section)
    - Only one thread at a time can execute in the critical region
    - All other threads are forced to wait on entry
    - When a thread leaves a critical region, another can enter
    - Example: sharing your bathroom with housemates

Mechanisms for Building Critical Regions

- To ensure that no two processes are ever in their critical regions at the same time.
  - Atomic read/write
    - Alternation
  - Locks
    - Primitive, minimal semantics, used to build others
  - Semaphores
    - Basic, easy to get the hang of, but hard to program with
  - Monitors
    - High-level, requires language support, operations implicit
  - Message Passing
    - System calls to send and receive messages between processes

Memory Management

Memory Management

- To provide a convenient abstraction for programming
- To allocate scarce memory resources among competing processes to maximize performance with minimal overhead
- To utilise concepts of physical and virtual memory to increase available memory
- **Memory hierarchy**
  - a few megabytes of very fast, expensive, volatile cache memory
  - a few gigabytes of medium-speed, medium-priced, volatile main memory
  - a few terabytes of slow, cheap, non-volatile magnetic or solid-state disk storage
- **Memory manager**
  - part of the operating system that manages (part of) the memory hierarchy

Virtual Memory

- Need to run programs that are too large to fit in memory
- Need to have systems that can support multiple programs running
- Swapping is not an attractive option because of speed
- Virtual memory (VM) is abstraction that the OS will provide for managing memory
  - Enables a program to execute with less than its complete data in physical memory
  - Many programs do not need all of their code and data at once (or ever) – no need to allocate memory for it
- OS will adjust amount of memory allocated to a process based upon its behaviour
- VM requires hardware support and OS management algorithms to pull it off

Virtual Memory Addresses

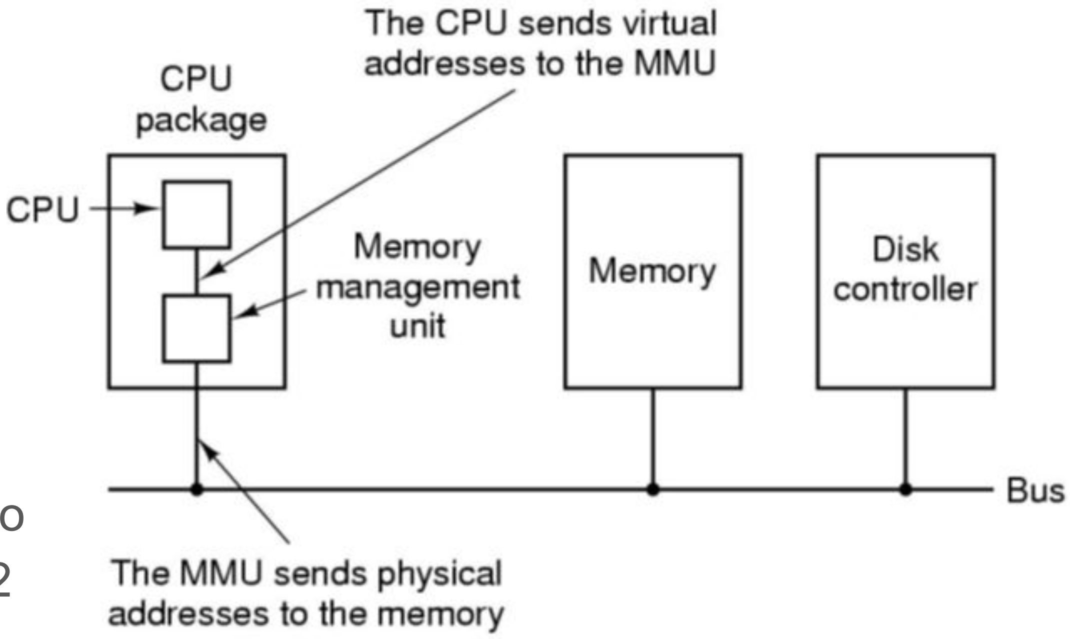
- Virtual addresses are independent of the actual physical location of the data referenced
- OS determines location of data in physical memory
- Instructions executed by the CPU issue virtual addresses
- Virtual addresses are translated by hardware into physical addresses (with help from OS)
- The set of virtual addresses that can be used by a process comprises its virtual address space

Memory Management Techniques

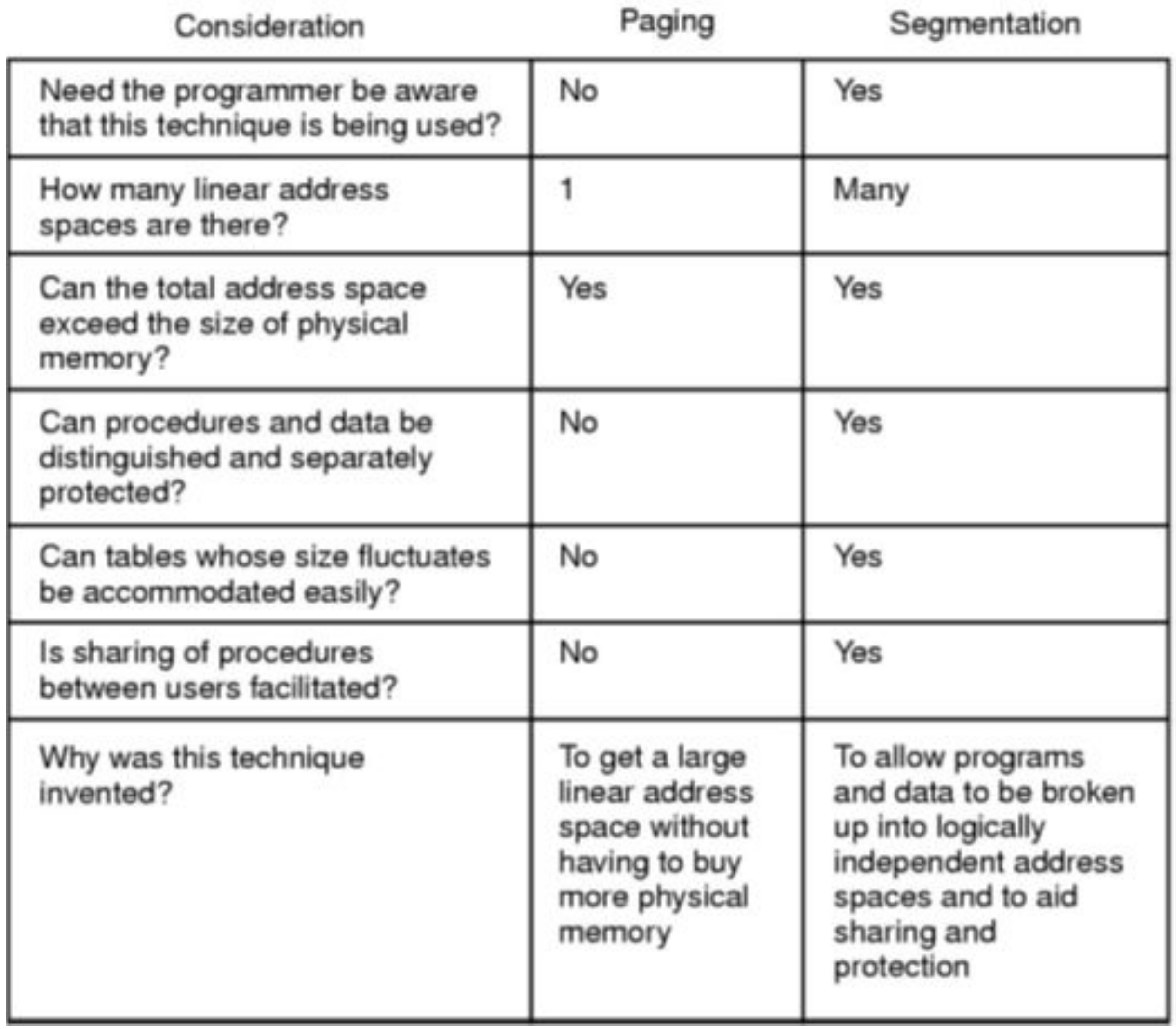
- Fixed partitions
  - easy to use, but internal fragmentation
- Variable partitions
  - more efficient, but external fragmentation
- Paging
  - use small, fixed size chunks, efficient for OS
- Segmentation
  - manage in chunks from user's perspective
- Combine paging and segmentation to get benefits of both

Memory Management Unit (MMU)

- When the program tries to access address 0, virtual address 0 is sent to the MMU.
- The MMU sees that this virtual address falls in page 0 (0 to 4095), which according to its mapping is page frame 2 (8192 to 12287).
- It thus transforms the address to 8192 and outputs address 8192 onto the bus.


The CPU sends virtual addresses to the MMU
CPU package
CPU
Memory management unit
Memory
Disk controller
Bus
The MMU sends physical addresses to the memory

Segmentation and Paging



Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection