Comp 104: Operating Systems Concepts

Processes

Management

Scheduling & Resource Allocation

1

Today

- OS evolution
- Introduction to processes
- OS structure

2

Evolution of OS

- Largely driven by desire to do something useful when a program cannot continue (maximise throughput)
- Early systems:
  - 'Job' loaded from punched cards or tape, output to printer
  - Job may include loading compiler, assembler, linker, data etc.
  - CPU idle for much of the time
- Batch systems:
  - Job passed to human operator
  - Operator groups jobs into batches with similar characteristics, e.g. all programs using same compiler
  - More efficient use of resources

3

Multiprogramming

- Load several programs into memory simultaneously, all sharing single CPU
- When running program cannot continue (e.g. waiting for I/O), switch to another
- Hence, I/O and computation overlap

4

Multi-Access (Time-Sharing)

- An extension of multiprogramming
- CPU is switched rapidly between processes to give illusion of uninterrupted execution in parallel (multitasking)
  - users can interact with programs
  - users see their own 'virtual machine'
  - resources (printers, disks etc.) are shared, but this is largely transparent

5

Question

- The following two statements describe the performance of two programs (where the computation and input/output could be interleaved):
  - A performs a total of 20 seconds of computation and 15 seconds of input/output.
  - B performs a total of 30 seconds of computation and 10 seconds of I/O
- Which of the following are true?
  - I. It will take up to 50 seconds to run A and B sequentially
  - II. It will take up to 75 seconds to run A and B sequentially
  - III. Using multiprogramming, the shortest time to execute both is 50 seconds
  - IV. Using multiprogramming, the shortest time to execute both is 40 seconds

a) I and III
b) I and IV
c) II and III
d) II and IV
e) None of the above

Answer: c
•If run sequentially, A needs to finish before B can begin, therefore II is true.
•With multiprogramming, I/O for one process can take pave whilst the computation takes place for another. Therefore III is true

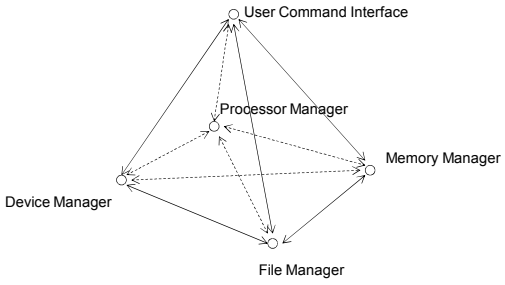
6

Implications

- Need to decide which programs to load from disk into memory (job scheduling)
- Need to decide which program to execute next (CPU scheduling)
- Consider disk space as extension of main memory (virtual memory)
- Memory allocation
- Disk/file allocation
- Protection/security

7

Operating System – An Abstract View



8

Processes

- A *program* is a representation of an algorithm in some programming language; i.e. it is *static*
- A *process* refers to the activity performed by a computer when executing a program; i.e. it is *dynamic*
- A process is created when a program or command is executed

9

Process Characteristics

- Process characteristics:
  - Requires space in memory where it resides during execution
  - During its execution it may require other resources such as data files or I/O
  - It passes through several states from its initial creation to its completion within the computer system (more details on these states to come in later lectures)

10

Processes

- A process needs resources, such as CPU time, memory, files and I/O devices, to accomplish its task.
- These resources are allocated either when the program is created, or when it is executing.
- Operating-system processes execute system code and user-processes execute user code
  - All these processes could potentially execute concurrently

11

Processes

- The Processor Manager is responsible for overseeing the following activities in relation to process management:
  - Creation and deletion of both system and user processes
  - Scheduling processes
  - Provision of mechanisms for synchronisation and communication of processes
  - Deadlock handling for processes

12

O.S. Structure

- Often consists of:
  - A central nucleus or kernel
    - resides permanently in memory
    - performs low-level, frequently needed activity
  - A set of processes
    - may be system level or user level
  - processes interact with kernel via system calls
    - e.g. create process, run program, open file
  - kernel and system level processes may operate in privileged mode

13

Command Interpreter

- Accepts and runs commands specified by user
  - Hence provides user's view of OS
- May be graphical, e.g. Windows
- May be textual, e.g. UNIX shell
  - bash, ksh, csh
  - Some commands built into shell, others loaded from separate executable files
  - Shell also has sophisticated control structures such as loops, if-statements and procedures

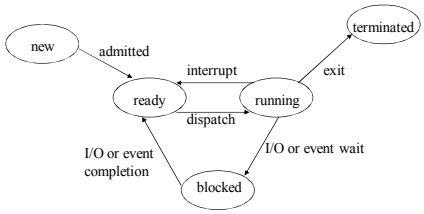
14

Process States

- Running
  - on a uniprocessor machine, only one process can be executing at any time
  - may be interrupted at end of time-slice if no I/O requests or system calls performed
- Ready
  - refers to a process that is able to run, but does not currently have the CPU
- Waiting(Blocked)
  - refers to a process that is unable to continue, even if granted the CPU

15

State Changes



16

Question

- A running process makes a system call to read data from a file. Which process state should it enter next?

a) New
b) Ready
c) Running
d) Blocked
e) Terminated

Answer: d
Blocked; it may take some time before the file system can read the file (e.g. on a networked file store), so the process is blocked until the data is available.

17

Process Descriptors

- For each process, the OS kernel maintainers a descriptor or Process Control Block (PCB)
- PCB contains info like
  - unique process ID
  - user ID of process owner
  - process state
  - position in memory
  - accounting stats. (time used etc.)
  - resources allocated (open files, devices, etc.)
  - register values (process counter, etc)

18

Context Switch

- When a process is interrupted
  - all current state information (including program counter and other registers) is saved into PCB
  - PCB is put into a queue
    - may have several, e.g. for different devices
  - the kernel may do some of its own work
    - e.g. handling a system call
  - the PCB of a process from the ready queue is selected, and its context restored
- Whole context switch is an expensive overhead
  - hardware support may help
    - e.g. multiple register sets

19

PCBs and Queuing

- The PCB of each process is updated as the process progresses from the start to the end of its execution
- Queues use PCBs to track the processes' progress through the system. The PCBs are linked to form queues:
  - 'Ready queue' linking the PCBs for every 'ready' process
  - 'New queue' linking the PCBs for processes just entering the system

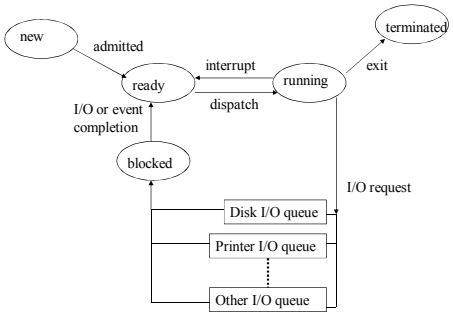
20

PCBs and Queuing

- Processes that are 'blocked' are linked together by 'reason for waiting'
  - PCBs for these processes are linked into several queues
    - e.g. those waiting for I/O on a specific disk drive are linked together, those waiting for a printer are linked in a different queue
- All queues need to be effectively managed in an order that is determined by the process scheduling policies and algorithms

21

Queuing



22

Inter-Process Communication

- Inter-Process Communication (IPC) mechanisms allow processes to talk to each other
- IPC useful when processes working together (cooperating processes)
  - synchronisation and/or passing data
- For example in UNIX:
  - signals
  - pipes
  - sockets

23

Signals

- A process can usually be terminated by typing CTRL-C
  - Actually sends a signal to process
  - Process responds by aborting
- Signals can be sent from one process to another
  - signal() system call
- Signals can be sent from the command line using kill command
  - Format: kill -<signal> <pid>
  - e.g. kill -9 12345 sends signal 9 (kill signal) to process 12345

24

Responding to Signals

- A receiving process can respond to a signal in three ways:
  - Perform default action (e.g. abort)
  - Ignore the signal
  - 'Catch' the signal; i.e. execute a designated procedure
- The 'kill' signal (signal 9) cannot be caught or ignored
  - Guaranteed way to stop process

Example kill signals

1 HUP (hang up)
2 INT (interrupt)
3 QUIT (quit)
6 ABRT (abort)
9 KILL (non-catchable, non-ignorable kill)
14 ALRM (alarm clock)
15 TERM (software termination signal)

25

Pipes

- The UNIX command 'wc –l file' counts the number of lines in file
- If we just type 'wc –l' we don't get an error
- Instead, data is read from standard input (keyboard by default)
  - Similarly for output files and standard output (screen)
- The pipe symbol '|' attaches the standard output of one program to the standard input of another, e.g. who | wc -l

Common wc flags

-l number of lines
-w number of words
-c number of characters

By default, all three stats are displayed. Flags state what stats appear…

26

Client-Server Examples

- The following are examples of common servers:
  - Web server: accessed by client's web browser
  - Mail server: retrieving and sending emails to clients
  - File server: holding documents to be accessed by clients
  - Database server: providing database services to clients, e.g. customer database, stock database…
  - etc

27