Lecture Objectives

1. How do we make best use of the processor cycles?

2. How do we do Multiprogramming?

3. What is a Process?

4. How are they managed?
Why have Processes?

- Computers (were) expensive
- CPU idle while waiting for I/O (polled)
- Could have several programs in memory
- Switch to another as they engage in I/O
- Now important management concept

In More Detail
**A Process**

- A Process is a running program - the program that the CPU is currently fetching and executing instructions for.
- A process exists in a virtual machine consisting of:
  - memory
  - a schedule
  - a set of I/O channels that it is allowed to use.

**Process Life Cycle**

- To start a program, the OS must:
  - Find the binary/executable on disk
  - Allocate storage for the code and data
  - Allocate swap space on disk
  - Map the pages in memory and swap space
  - Copy the code into the physical pages
  - Start execution by saving current context (see later) and jumping (loading the PC) to the start address of the code!
## Important Features

- Code and data for process must be in memory
- When process is suspended from running need to save its **current state**
- Saving, restoring this state and deciding which process to run is known as **context switching** – “wasted” time or **overhead**
- Needs to be **policy** for choosing next process to run - high level **scheduler**

## Process State Transitions

- **Running**
  - I/O Request
- **Ready to Run**
  - Time Expired
  - Dispatch
- **Blocked**
  - I/O Complete
Process Representation

1. OS keeps Process Control Block (PCB) for each process.
2. Contains management information:
   - Location of P in memory
   - P’s state
   - P’s owner (user)
   - etc.

Memory

Kernel Space (OS)

User Space

PCB1

PCB2

PCB3

PCB4

PCB0

PCB1

PCB2

PCB3

PCB4

Current (Running) P: PCB5

Ready to Run: PCB3 → PCB1 → PCB0

Blocked: PCB2 → PCB4

Note: Idle “Process”
Process State Transitions - 2

- **Running->Ready**
  - Time slot expires
  - Scheduler determines time slot

- **Ready->Running**
  - Current P-> Blocked or *pre-empted*
  - Highest priority runnable-> running
  - Null/Idle process if Q empty

- **Running->Blocked**
  - P makes *synchronous I/O request*
  - or waits for earlier *asynchronous I/O request*

- **Blocked->Ready**
  - I/O or events complete

---

Process State Transitions - 3

- **Null/Idle Process**
  - May do something useful: machine diagnostics, compute prime numbers, etc
  - or just wait in kernel loop for event (I/O)

- **Scheduling vs Dispatching**
  - Dispatcher just takes next P from Q and sets it running
  - Scheduler decides on ordering of Ps on Q on basis of policy, eg, shortest job, premium payment, real-time activity, etc.
Processes are defined by their code, data, and state.

What does the state consist of:
- Program Counter
- Stack Pointer
- Register Contents
- Page/MMU register Contents
- Anything else?

How do we suspend one program, to start another (i.e. give the illusion of concurrency)?
**Process Control Block**

Contains **state information** such as:

- **Process State**: Ready, etc
- **Process ID**: to relate I/O, events inter-process communications (IPC), etc
- **Priority**: or time slot, time to run etc.
- **Memory pointers**: location of process code and data
- **Resources Allocated**: terminals, devices
- **Register Save Area**: processor registers, stack pointer etc.
- **Owner**: User ID
- **Parent**: Parent process ID
- **Owner**: Parent

---

**Process Control Block - 2**

When P **swapped out**:

- Not all of PCB needs to be retained in kernel
- Need to also store swap device address of Process image in PCB
**Cost of Context Switch?**

- A context switch entails at least 1 trap (call to scheduler or I/O or clock interrupt)
- Then it involves a **full save of registers and page table**
- What if we knew we were only going to do a bit of work? Couldn’t we do something quicker?
- Yes: threads......

---

**Threads - bare processes...**

- Threads are also known as **“lightweight processes”**
- Assume we have a “safe” language, then we can have “concurrency” and scheduler, but avoid saving all registers and so on - let threads run in **same virtual address space and so on**
- All the same except that scheduler is part of library not a system call (cheaper) and much less state saved and restored in a context switch (minimum is stack, register and PC)
Scheduling Algorithms

- What is the order in which a scheduler chooses the next process when another one suspends or stops?
  - Random, round robin (fair) priority, mix
- Can try to balance between I/O and CPU intensive based on past behaviour
- Can have a real-time requirement (e.g. must respond before the reactor blows up).
- Complex- especially in newer multiprocessor or distributed systems!

Communicating Processes

- Normally in a time-sharing multi-tasking system processes are unrelated. But, sometimes...we want them to communicate with each other.
- Communicating processes in different machines or address spaces use messages and protocols.
- Communicating processes in the same address space can share memory, and therefore can communicate by sharing variables (memory)
### Shared Variable Comms

- There are *interesting* problems with concurrent access to shared variables:
  - Run process A, which reads bank account \( v \)
  - Switch to process B, which reads same value
  - Switch to process A which adds £100 to it and writes it
  - Switch to process B which adds £100 to it and writes it
  - Result is wrong (only 1 update had an effect)
  - "Lost update" problem

### Lost Update Problem

- Result should be £273
- One update lost because balance read a second time before first update complete
- Need to *bind* two steps of update together
- A could *lock* data until it has finished with it
Shared Variable Comms - 2

- Solutions include:
  - Locks
  - Queues
  - Monitors - lots of others - will be covered later in course - a difficult subject

Problems with message passing

- Concurrent systems that pass messages or share variables can *deadlock*
  - Process a is waiting for process c
  - Process c is waiting for b
  - Process b is waiting for a
  - (consider problems that can occur at a road junction when each car blocks the exit of another)
- or *livelock* or be unfair and so on
Summary

- Process - running program
- Scheduler picks between processes to run
- Scheduler saves and restores contexts in process table
- Virtual Machine hides processes from each other
- Shared memory and messages can allow communication between processes
- Concurrency causes problems

Further Reading

- and many more