ELEC 377 – Operating Systems

Week 2 – Class 1

Last Week

- Computer System Structure
- Interrupts, Traps, I/O Queues
- C language
- Storage Structure (Main Memory, Secondary, Cache)
- Hardware Protection – System Modes
- Operating System is interrupt driven
- System Calls

Labs

- Lab 1 is on the Website, prelab is due at beginning of lab

What is a process?

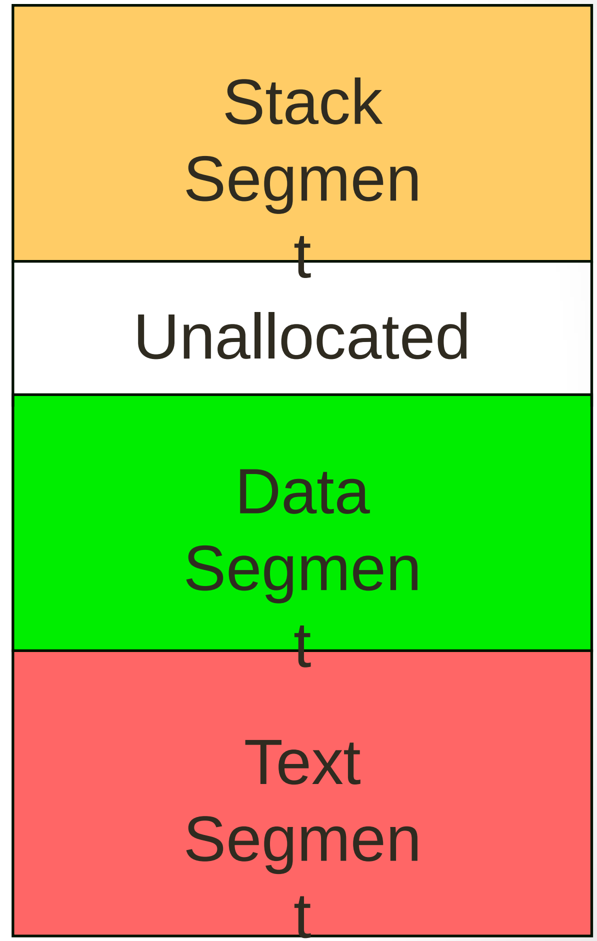
- An operating system handles a variety of programs in a variety of ways
  - ◊ Batch systems handle jobs
  - ◊ Timesharing systems calls them user programs or tasks.
- Job and process are used almost interchangeably.
- A process is a program in execution, and all of the resources associated with that executing instance of the program
  - ◊ Memory
  - ◊ Program Counter
  - ◊ Open files, other devices, etc.

Program Layout

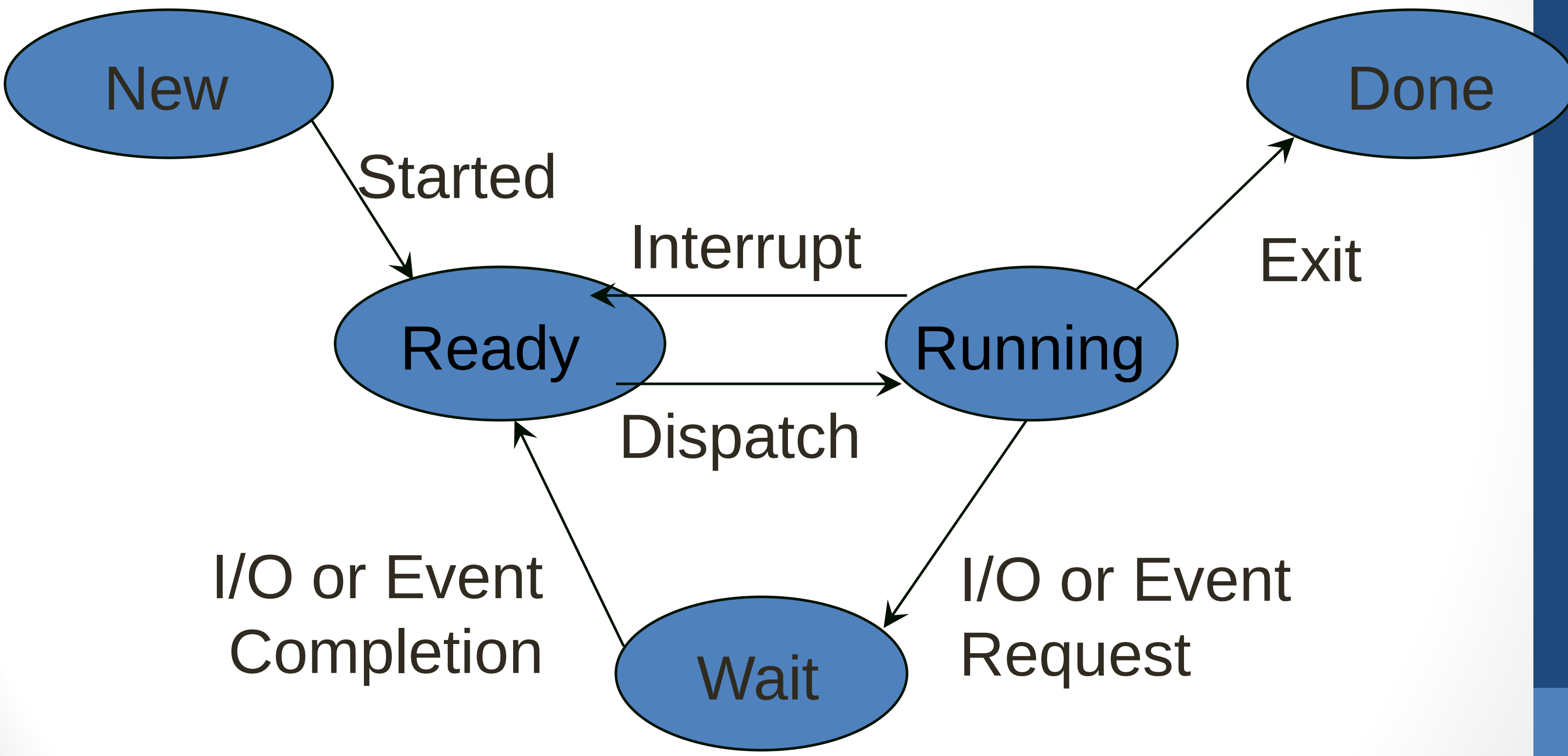
Procedure Call/Return
Local Variables
– grows downwards

Top Half is the Heap (malloc)
– grows upwards
Bottom Half is Global Vars

Executable Code
Binary Machine Instructions
Usually Shared



Process State



Process Control Block

- One allocated for each process and sometimes for each thread
- Repository for information that varies from process to process
- Some operating systems have a pre allocated number of them. i.e. an array (early UNIX)
- Some permit dynamic allocation (Amiga OS)

Queue Info
Process State
Schedule Info
Process Id
Program Counter
Registers
Memory Info
Open File, etc.
Accounting Info

Context Switch

- Depends on hardware

◊ number of registers to save/restore

◊ supporting hardware (memory management, etc).

◊ hardware support

  – banks of registers

  – multiple supporting hardware (cache, TLB, etc.)

- System is not doing useful work

◊ overhead of multitasking

◊ 1 to 1000 micro seconds

Scheduling Queues

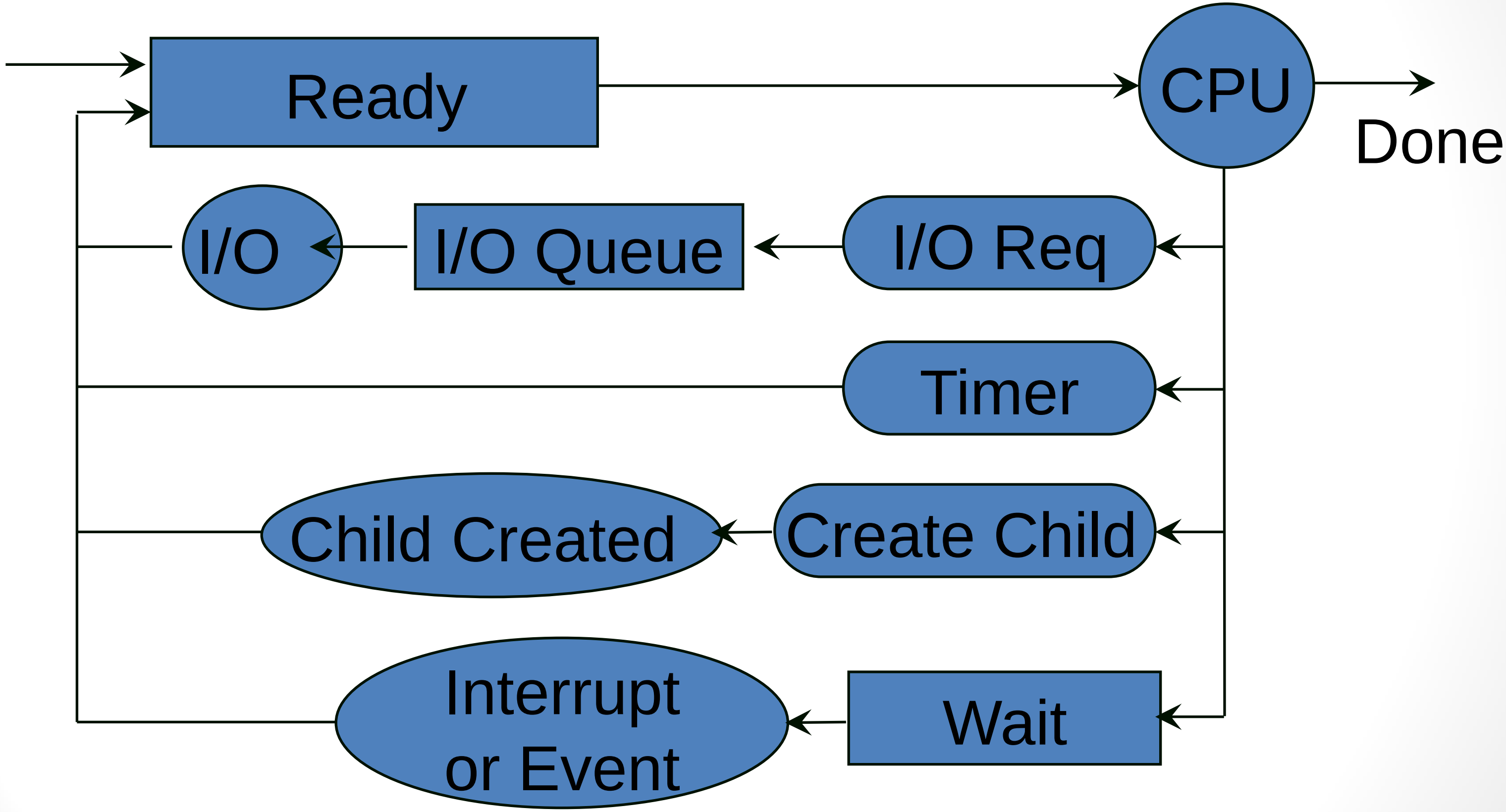
- Process may be on more than one queue at a time
- Job Queue (all jobs)

◊ easy way to find status of any particular job

- Ready Queue - all process ready to run
- Device Queues - all processes waiting for I/O
- Suspend Queue – processes that have been suspended.

- As processes change state, issue system calls, are interrupted, they move between the queues.
  ◊ queues are used to keep track of the processes

Scheduling Queues



Types of Processes

- I/O Bound (more time in I/O)
  - ◊ Mostly I/O
  - ◊ Interactive processes (editors, chat, web)
  - ◊ High priority
- CPU bound(more time in Computations)
  - ◊ A lot of computation
  - ◊ Scientific programs
  - ◊ 3D games
  - ◊ Low priority
- Many programs change over time
  - ◊ Periods where I/O bound - reading/writing data
  - ◊ Periods of CPU bound - calculating
  - ◊ Compilers

Scheduling

- Scheduler has to choose the next job to run
  - ◊ short term scheduler (into CPU)
  - ◊ process chosen will last until interrupt, system call or timer expires (short term)
  - ◊ simple, low penalty for mistake
- In batch systems, long term scheduler chooses which job to load into memory from the job pool
  - ◊ longer term decision, high penalty for mistake
  - ◊ more sophisticated algorithms
  - ◊ controls balance of multiprogramming

Medium Term Scheduler

- In interactive systems (Unix, Windows, MacOS), medium term scheduler to swap inactive tasks in and out of memory

◊ partially executed processes may be removed from memory and later restored

◊ In older unix systems (i.e. PDP-11), may be necessary to increase size of memory partition for process.

Process Creation

- Almost as many ways to create a process as there are operating systems!!
- Usually a parent–child relationship
- Parent process creates one or more child processes
- Parent usually has some sort of control over child
- Unix - first user level process is called *init*
  - ◊ starts daemon processes (network servers, printer servers, etc.)
  - ◊ terminal login processes
- Terminal login processes create shell processes when users log in
- Shells create user processes as children

Process Creation

- Resource Sharing

◊ Parent and children share all resources

◊ Parent partitions resources to children (subset)

◊ Parent and child share no resources

- Execution possibilities

◊ parent is suspended and waits for children (DOS)

◊ parent continues to execute concurrently with children (UNIX)

◊ some operating systems have both as separate system calls.

Process Creation

- Address possibilities
- ◊ Children and Parent share address space (threads, linux clone())
- ◊ Child is copy (including open files) [UNIX]
- ◊ Child starts new program specified in system call [VMS]
- ◊ Some operating systems have more than one of these.

Process Termination

- Process asks the operating system to exit or abort
- ◊ Parent informed of exit status
- ◊ process resources are recovered by operating system (otherwise a leak!!)
- Child may make an error (memory, privilege, etc.)
- ◊ OS aborts and informs parent
- Parent may abort the child
- ◊ Task assigned to child is no longer required (network service for example)
- ◊ Parent is exiting
  – Some operating systems auto kill children, and children's children
  – some permit children to be inherited by parent of parent [UNIX].

Cooperating Processes

- Independent process cannot affect or be affected by another process
- Cooperating processes can affect or be affected by other processes
- Why provide cooperating processes?

Example: Producer Consumer

- Producer generates information that is passed to a consumer processes

- Unbounded buffer (assumes no bounds on size of buffer used to share data)

◊ Only the consumer needs to wait

- Bounded buffer – there is a limit

◊ both produce and consumer may have to wait

- Example of Shared Memory Interprocess communication

Interprocess Communication

- Mechanism for processes to communicate and to synchronize their actions

- Message System

◊ send(message), receive(message)

◊ need to establish a connection between processes

Implementation Questions

- How are links established?
- Number of processes?
- Link Capacity
- Max Message Size (variable)
- Unidirectional/Bidirectional
- Message copied or pointer sent?
- Explicit/Implicit buffering

Direct Communication

- Processes explicitly identify each other

◊ send(P,message)

◊ receive(Q, &message)

- addressing may be asymmetric

◊ send(P,message)

◊ receive(&id,&message)

Direct Communication

- Advantages?
  - Explicit
  - Simple
- Disadvantages?
- Limited modularity of the resulting process definitions
- Changing an ID of a process -> may need to examine all other process definitions.

Indirect Communication

- Mailboxes
- ◊ each mailbox has a unique id
- ◊ processes share the mailbox
- What if more than one process wants to receive a message from a mailbox
- ◊ Only allow one process to read mailbox
- ◊ First come – first serve
- ◊ Multiple receivers

Indirect Communication

- Advantages?
  - More flexible
  - No question who received the message from a mailbox
- Disadvantages?
  - Operating system must provide mailbox mechanism
  - Mailbox ownership may be passed -> could result in multiple receivers.

Synchronization

- Coordination between sending and receiving process
  - ◊ blocking vs non-blocking
  - ◊ applies to both sender and receiver
  - ◊ blocking is synchronous
  - ◊ non-blocking is asynchronous

- if both send and receive are blocking -> rendezvous
  - ◊ Ada

Buffering

- Zero Capacity -> rendezvous
  - Queue of length 0
  - Sender blocks until message is received

- Bounded -> sender may need to wait or abandon the message
  - If queue is full, sender blocks.

- Unbounded -> sender never needs to wait.

◊ resource intensive

◊ non-guaranteed delivery (IP/UDP)