Today ...

- CPU scheduling
- Starting memory management

So far ...

- SJF has short wait times, but is unfair
- FCFS is fair, but can lead to long wait times
  - Short jobs can get stuck behind long jobs

How can we be fair, but avoid the “short jobs getting stuck behind long jobs” problem?

**IDEA:** Allow preemption
- Don’t always run jobs to completion.
- Preempt a job that’s running too long

Algorithm: Round Robin

- Designed for time-sharing systems
- Pre-emptive
- Ready queue is circular
  - Each process is allowed to run for time quantum $q$ before being preempted and put back on queue
- Choice of quantum (aka time slice) is critical
  - as $q \to \infty$, RR $\to$ FCFS;
  - as $q \to 0$, RR $\to$ processor sharing (PS)
- we want $q$ to be large w.r.t. the context switch time

Priority Scheduling

- A priority, $p$, is associated with each process
- Highest priority job is selected from Ready queue
  - Can be pre-emptive or non-preemptive
- Enforcing this policy is tricky
  - A low priority task may never get to run (starvation)
  - A low priority task may prevent a high priority task from making progress by holding a resource (priority inversion)

Priority Inversion Example

- Mars Rover Pathfinder bug
  - In the press:
    - “software glitches”,
    - “computer was doing too many things at once”

<table>
<thead>
<tr>
<th>HIGH priority</th>
<th>LOW priority</th>
<th>MED priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus management</td>
<td>data gathering</td>
<td>communications task</td>
</tr>
<tr>
<td>(does not use bus)</td>
<td>(exclusive use)</td>
<td>(exclusive use)</td>
</tr>
</tbody>
</table>

Shared memory
Priority Inversion Example

- Mars Rover Pathfinder bug
  - Shared "information bus" – essentially shared memory area
  - Mutual exclusion provided by lock on info bus
  - High priority "bus management" task moves data in/out of information bus
  - Low priority "data gathering" task writes data to the information bus
  - Medium priority, compute-bound "communications" task that does not use the information bus

See the problem?
- Data gathering task locks bus and is preempted by higher priority bus management task, which blocks on the lock. If communications task becomes runnable, data gathering task can't complete and release the lock so high priority task stays blocked.

What do real systems do?

- Combination of
  - Multi-level queue scheduling
    - Typically with RR and priorities
  - Feedback scheduling

What does that mean …?

Multi-Level Queue Scheduling

- Have multiple ready queues
  - Each runnable process is on only one queue
- Processes are assigned to a queue
  - Criteria include job class, priority, etc.
  - Each queue has its own scheduling algorithm
  - Another level of scheduling decides which queue to choose next
  - Usually priority-based

Feedback Scheduling

- Motivation:
  - Want to give priority to shorter jobs
  - Want to give priority to IO bound jobs
  - Want to give priority to interactive jobs
  - Want to …

  - But don’t know beforehand whether a job is short or long and whether it's IO bound or CPU bound …

Feedback Scheduling

- Adjust criteria for choosing a particular process based on past history
  - Can prefer processes that do not use full quantum
  - Can change priority of processes based on age
  - Can change priority of processes based on CPU consumed so far
  - Can boost priority following a user-input event
- Combine with MLQ to move processes between queues
Linux 2.6 CPU scheduling

- Combination of
  - Multilevel queues
  - With priorities and RR
  - Feedback scheduling
- Distinguishes 3 classes
  - Realtime FIFO processes
  - Realtime RR processes
  - Timesharing processes
- Our discussion focuses on timesharing processes

How do you avoid starvation of low priority processes?

- How to determine priority of process?
  - Some processes are more important than others
    - Assign static priorities [-20, 19]
    - Done through nice system call
  - Interactive & I/O bound processes should be favored
    - Give bonus/penalty on top of static priority
    - priority = static_priority + bonus
  - How to identify interactive / I/O bound processes?
    - Maintain sleep_avg (time process is sleeping vs CPU time)

How to set timeslice and granularity?

- timeslice depends on static priority
  - Between 5ms and 800ms
- granularity depends on dynamic priority

Many of the details keep changing …
New topic

- Memory management

Memory Management

- Every active process needs memory
- CPU scheduling allows processes to share (multiplex) the processor
- Must figure out how to share main memory as well
- What should our goals be?
  - Support enough active processes to keep CPU busy
  - Use memory efficiently (minimize wasted memory)
  - Keep memory management overhead small
  - ... while satisfying five basic requirements

Requirements

- Relocation
  - Programmers don’t know what physical memory will be available when their programs run
  - Scheduler may swap processes in/out of memory, need to be able to bring it back in to a different region of memory
  - This implies we will need some type of address translation

- Logical Organization
  - Machine accesses.addresses memory as a one-dimensional array of bytes
  - Programmers organize code in modules
  - Need to map between these views

More requirements

- Protection
  - A process’s memory should be protected from unwanted access by other processes, both intentional and accidental
  - Requires hardware support

- Sharing
  - In some instances, processes need to be able to access the same memory
  - Need ways to specify and control what sharing is allowed

- Physical Organization
  - Memory and Disk form a two-level hierarchy, flow of information between levels must be managed
  - CPU can only access data in registers or memory, not disk

Meeting the requirements

- Modern systems use virtual memory
  - Complicated technique requiring hardware & software support
- We’ll build up to virtual memory by looking at some simpler schemes first
  - Fixed partitioning
  - Dynamic partitioning
  - Paging
  - Segmentation
- We’ll begin with loading and address translation

Address Binding

- Programs must be in memory to execute
  - Program binary is loaded into a process
  - Needs memory for code (instructions) & data
  - Addresses in program must be translated to real addresses
  - Programmers use symbolic addresses (i.e., variable names) to refer to memory locations
  - CPU fetches from, and stores to, real memory addresses

When are addresses bound?
When are addresses bound?
- Compile time
  - Called **absolute code** since binary contains real addresses
  - Disadvantage?
    - Must know what memory process will use during compilation
    - No relocation is possible

```
int main()
    int y;
    y = random();
    printf("%d",y);
```

When are addresses bound?
- Load time
  - Compiler translates (binds) symbolic addresses to **logical, relocatable** addresses within compilation unit (source file)
  - Linker takes collection of object files and translates addresses to **logical, absolute** addresses within executable
  - Resolves references to symbols defined in other files/modules
  - Loader translates logical absolute addresses to **physical** addresses when program is loaded into memory
  - Disadvantage?
    - Programs can be loaded to different address when they start, but cannot be relocated later

```
int main()
    int y;
    y = random();
    printf("%d",y);
```

A better plan
- Bind addresses at execution time

```
main.c
    translators
    cc1
    translators
    link
    a.out
```

Memory management
- Two key problems:
  - How do you allocate physical memory for a process?
  - How do you map **logical to physical** addresses?

Assume for now:
- Entire address space of process must be in main memory
- A process uses a contiguous chunk of main memory

Assume for now:
- Entire address space of process must be in main memory
- A process uses a contiguous chunk of main memory
Fixed Partitioning
- Divide memory into regions with fixed boundaries
  - Can be equal-size or unequal-size
  - A single process can be loaded into each remaining partition
  - Example: 2 processes with 5M and 2M, respectively.

Disadvantages?
- Memory is wasted if process is smaller than partition (internal fragmentation)
- What about programs that are larger than partition?

Placement w/ Fixed Partitions
- Number of partitions determines number of active processes
  - If all partitions are occupied by waiting processes, swap some out, bring others in
  - Equal-sized partitions:
    - Process can be loaded into any available partition
  - Unequal-sized partitions:
    - Queue-per-partition, assign process to smallest partition in which it will fit
    - A process always runs in the same size of partition
    - Single queue, assign process to smallest available partition

Placement Example (Queue per partition)
- Process 1 and Process 2 fit in same partition. With smallest-partition policy, both must share 8M partition while 16M partition goes unused.

Dynamic Partitioning
- Partitions vary in length and number over time
  - When a process is brought in to memory, a partition of exactly the right size is created to hold it

Disadvantages?

More Dynamic Partitioning
- As processes come and go, “holes” are created
  - Some blocks may be too small for any process
    - This is called external fragmentation
  - OS may move processes around to create larger chunks of free space
    - E.g. if Process 3 were allocated immediately following Process 1, we would have a 25M free partition
    - This is called compaction
    - Requires processes to be relocatable

Tracking Memory Allocation
- Bitmaps
  - 1 bit per allocation unit
    - “0” == free, “1” == allocated
  - Advantage/Disadvantages?
    - Allocating a N-unit chunk requires scanning bitmap for sequence of N zero’s
    - Slow

Memory: 
Bitmap: 1110000011100111111000011
Tracking Allocation (2)

- Free lists
  - Maintain linked list of allocated and free segments
  - List needs memory too. Where do we store it?
- Implicit list
  - Each block has header that records size and status
    (allocated or free)
  - Searching for free block is linear in total number of blocks

Freeing Blocks

- Adjacent free blocks can be coalesced

```
int p = malloc(3);
...
free(p);
```

- Easier if all blocks end with a footer with size/status info (called boundary tag)

Placement Algorithms

- Compaction is time-consuming and not always possible
- We can reduce the need for it by being careful about how memory is allocated to processes over time
- Given multiple blocks of free memory of sufficient size, how should we choose which one to use?

```
Operating system
8M
Available
18M
Process 1 - 5M
Available - 7M
Process 3 - 2M
Process 4 - 6M
```

Where should we place process?

Comparing Placement Algs.

- First-fit
  - Simplest, and often fastest and most efficient
  - May leave many small fragments near start of memory that must be searched repeatedly
- Best-fit
  - Left-over fragments tend to be small (unusable)
  - In practice, similar storage utilization to first-fit
- Worst-fit
  - Not as good as best-fit or first-fit in practice
- Quick-fit
  - Great for fast allocation, generally harder to coalesce

Problems with Partitioning

- With fixed partitioning, internal fragmentation is big problem
  - Scheme is too inflexible
- With dynamic partitioning, external fragmentation and management of space are major problems
- Basic problem is that processes must be allocated to contiguous blocks of physical memory
  - Hard to figure out how to size these blocks given that processes are not all the same
- We’ll look now at paging as a solution
Paging

- Partition memory into equal, fixed-size chunks
  - These are called page frames or simply frames
- Divide processes’ memory into chunks of the same size
  - These are called pages
- Possible page frame sizes are restricted to powers of 2 to simplify translation

Example of Paging

Suppose a new process, D, arrives needing 3 frames of memory

- We can fit Process D into memory, even though we don’t have 3 contiguous frames available!

Example of Paging

Suppose a new process, D, arrives needing 3 frames of memory

- Is there fragmentation with paging?
  - External fragmentation is eliminated
  - Internal fragmentation is at most a part of one page per process