IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering

Lecture 16:
Midterm 1 Review

Basic Information

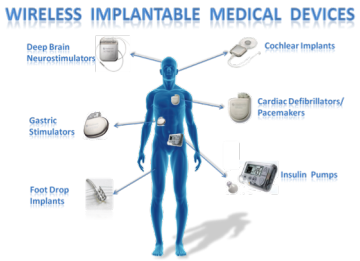
- Time
  - 09:00-09:50 am, Oct 4 (Friday)
- Location
  - Marston 2300
- Format
  - Similar to HW1
  - Closed book/notes
- Scope
  - Lecture 1 to today

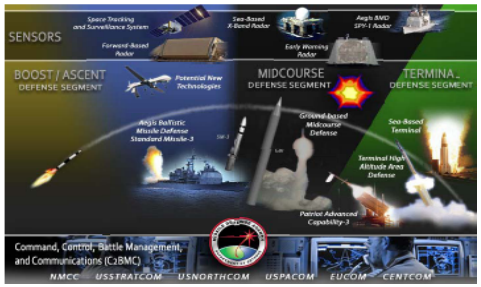
Review

- Introduction
- Process Management
  - Processes
  - Threads
  - Process/Thread Scheduling
  - Inter-Process Communication
  - Classic IPC Problems
  - Deadlocks

Review

- Introduction
  - Why OS
    - Fundamental to computer systems
      - Affects correctness, security, performance ... of entire system
    - Fundamental to modern society

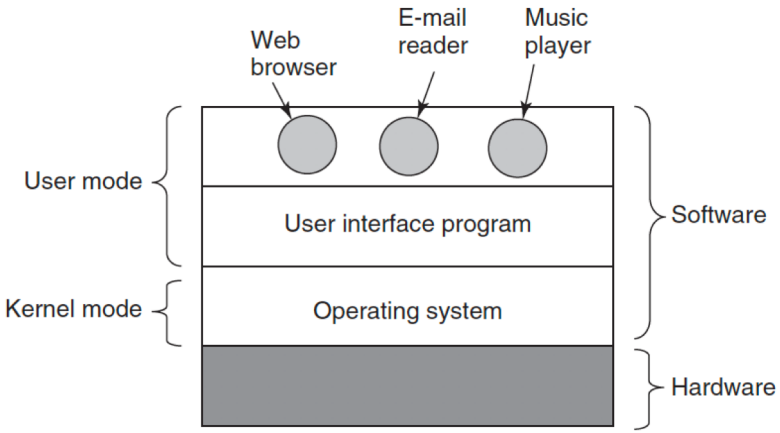


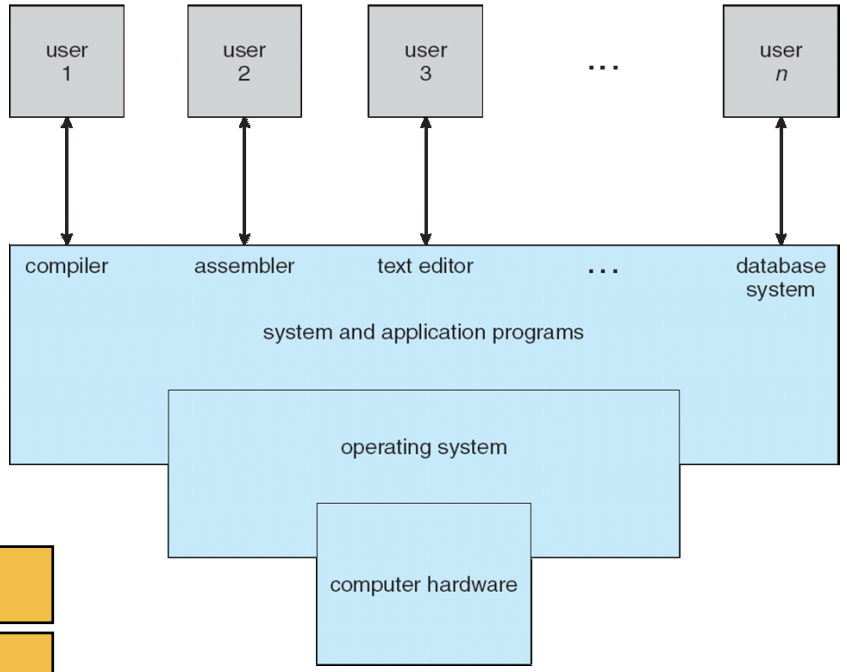


  - What's OS
    - OS is a resource manager
    - OS is a control program
      - an extended/virtualized machine with abstraction

Review

- Introduction
  - Computer System Structure





Users

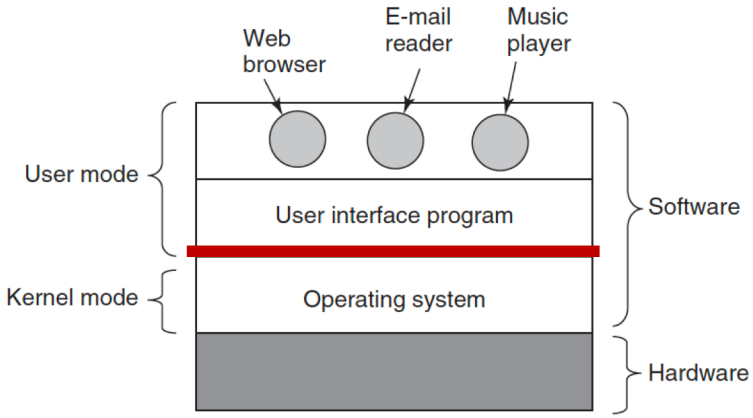
Applications

OS

Hardware

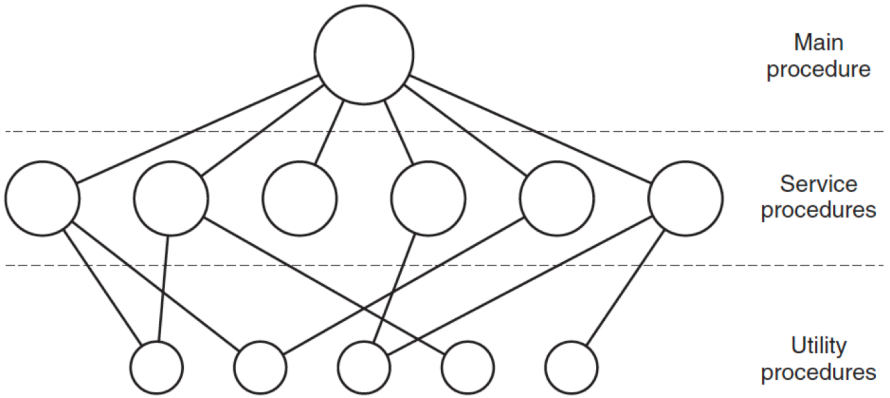
Review

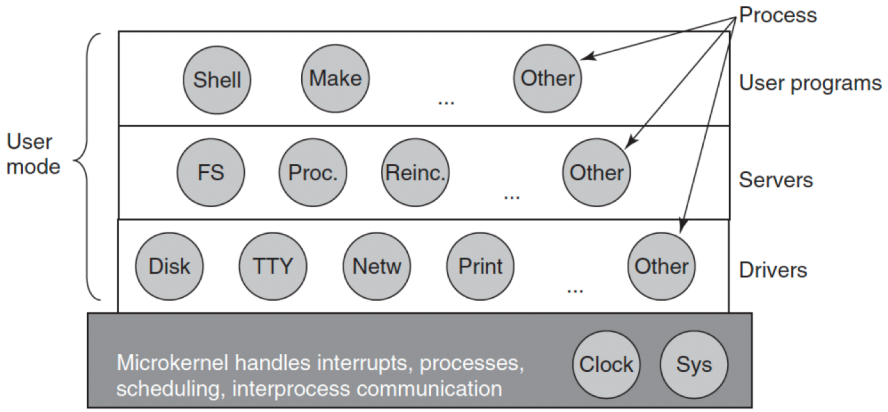
- Introduction
  - OS Abstractions for HW
    - CPU
      - process and/or thread
    - Memory
      - address space
    - Disks
      - Files
  - System Calls
    - Function calls implemented by OS
    - interface to apps/users



Review

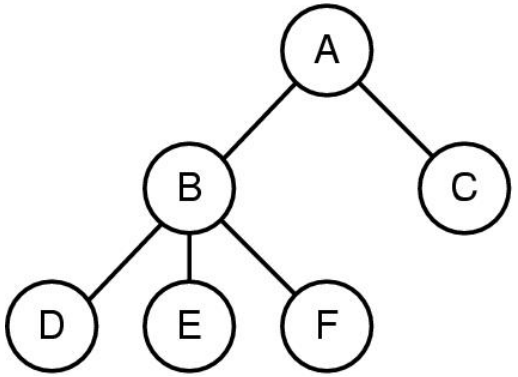
- Introduction
  - OS Structures
    - Monolithic kernel V.S. Microkernel
      - Monolithic: the OS runs as a single program in kernel mode
      - Microkernel: split the OS up into small, well-defined modules, only one of which—the microkernel—runs in kernel mode





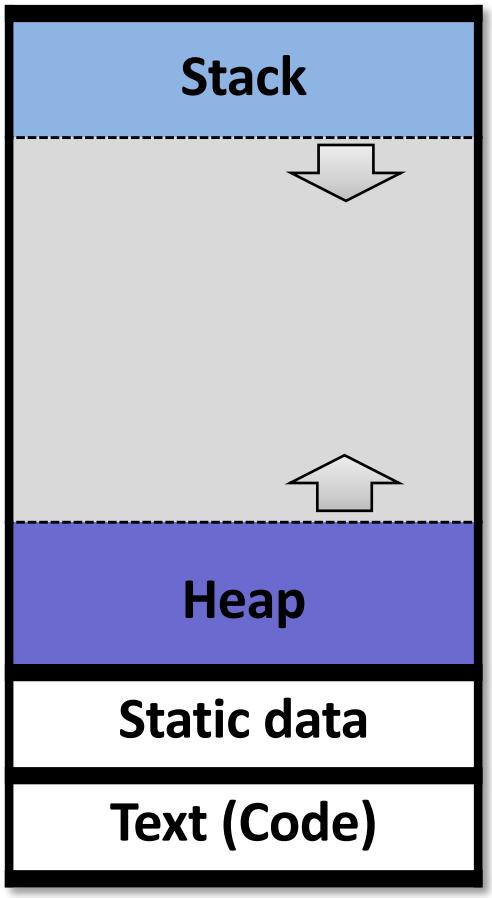
Review

- Process
  - A program *in execution*
  - holds all the info needed to run a program
    - Address space
      - Program (text), data, stack
    - Register values
      - Program counter, stack pointer, etc
- Process Tree
  - a hierarchy of related processes



Review

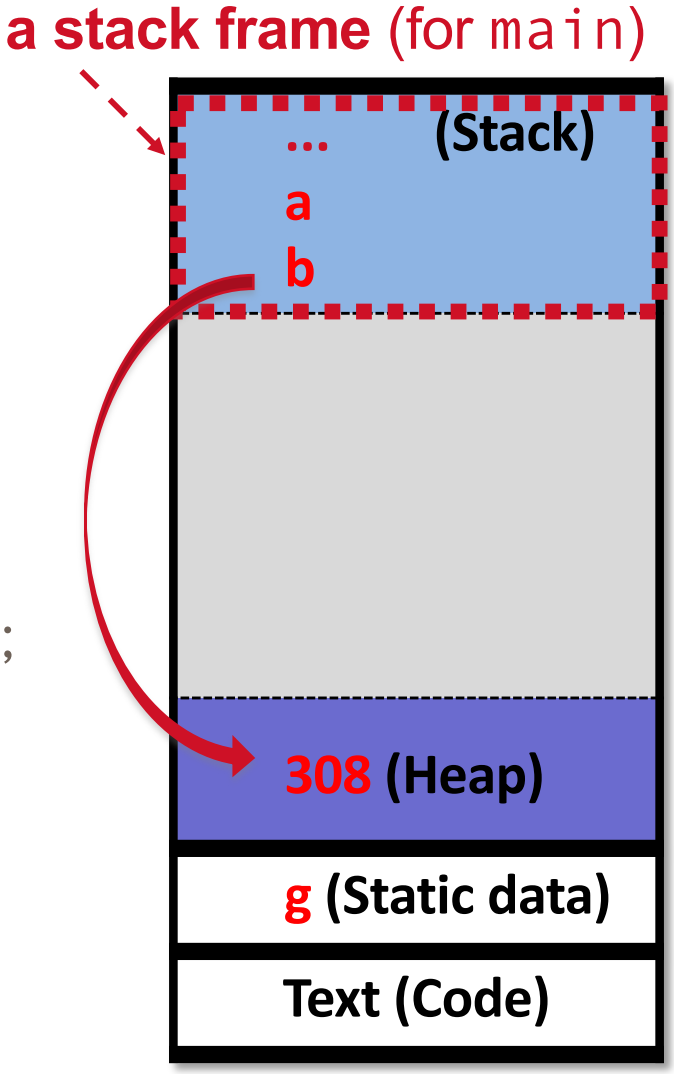
- Process Address Space
  - Four segments
    - stack
      - local variables, function parameters, return address
    - heap
      - dynamically allocated data
        - e.g., `malloc()`
    - static data
      - global/static variables
    - code (text)
      - instructions of the program



Review

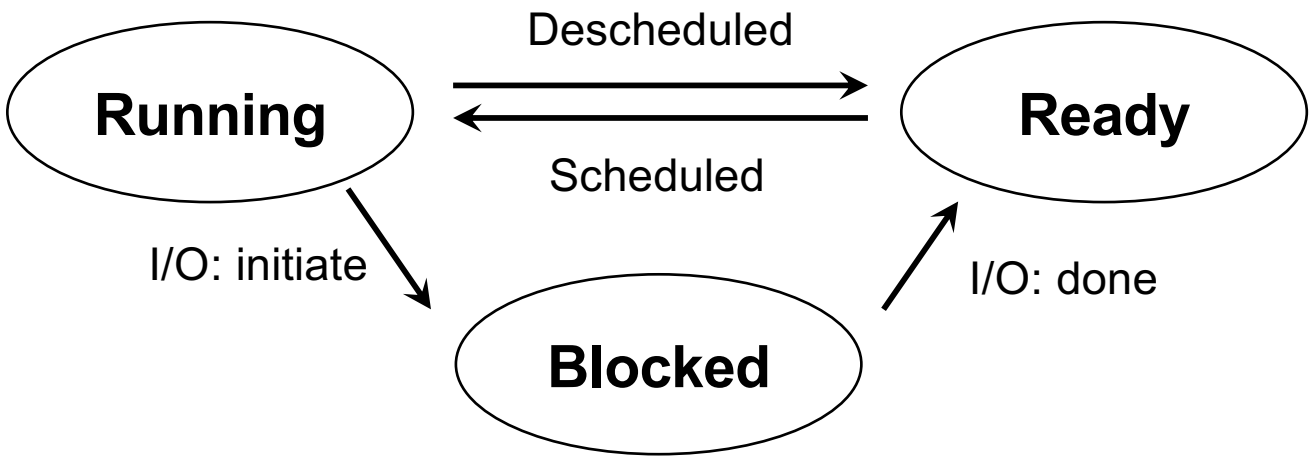
- Process Address Space
  - Example
    - Memory leak

```
int g;
int main() {
  int a;
  int*b = (int*)malloc(sizeof(int));
  *b = 308;
  return 0;
}
```



Review

- Process States
  - Three basic states
    - **Running**
    - **Ready**
    - **Blocked**
- State transition



Review

- Process Context
  - Stored in a “Process Control Block (PCB)”

Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children’s CPU time		
Time of next alarm		

- Context Switch
  - switching the CPU to another process by
    - saving the context of an old process
    - loading the context of a new process

Review

- Process APIs
  - Example: fork()

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid()); //get process ID
    int rc = fork();       // create a child process
    if (rc < 0) {          // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else {               // parent goes down this path (main)
        printf("hello, I am parent of %d (pid:%d)\n",
        rc, (int) getpid());
    }
    return 0;
}c
```

Review

- Process APIs
  - Example: fork()
    - results (non-deterministic)

```
prompt> ./p1
hello world (pid:29146)
hello, I am parent of 29147 (pid:29146)
hello, I am child (pid:29147)
prompt>
```

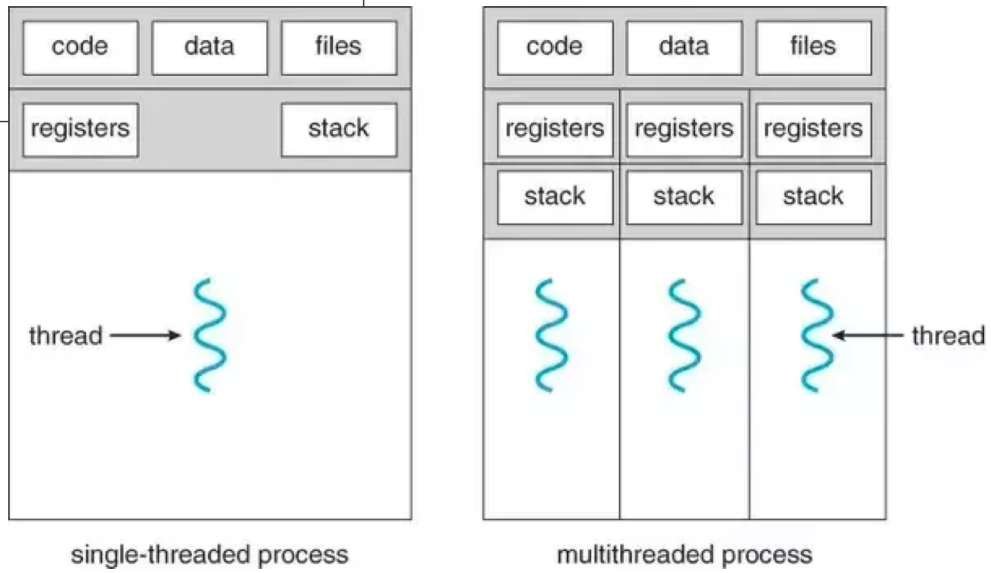
or

```
prompt> ./p1
hello world (pid:29146)
hello, I am child (pid:29147)
hello, I am parent of 29147 (pid:29146)
prompt>
```

Review

- Multi-threaded Process
  - Multiple threads of control within a process

Per-process items	Per-thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	



single-threaded process multithreaded process

Review

- Process/Thread Scheduling
  - First-Come, First-Served (FCFS)
  - Shortest-Job-First (SJF)
  - Shortest Remaining Time Next
  - Round Robin (RR)
  - Priority Scheduling

Review

- Process/Thread Scheduling
  - First-Come, First-Served (FCFS)
    - Scheduling based on the arrival order of processes

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Assume the processes arrive in the order: ***P_2 , P_3 , P_1***
- The Gantt Chart for the schedule is:

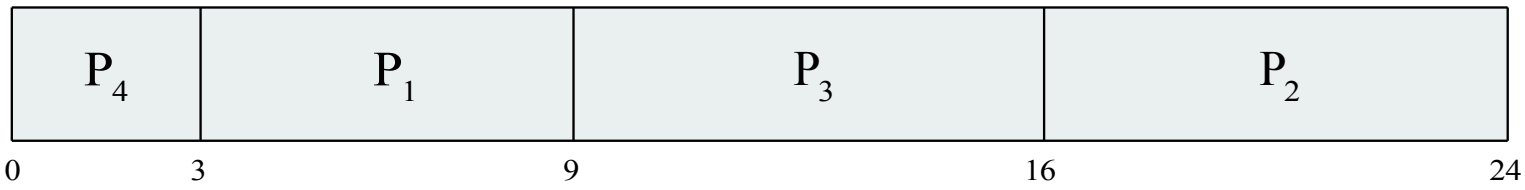
- Waiting time: P_1 = 6; P_2 = 0; P_3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3

Review

- Process/Thread Scheduling
  - Shortest-Job-First (SJF)
    - Scheduling based on (predicted) CPU burst time

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- Assume all processes are ready at time 0
- Gantt Chart:



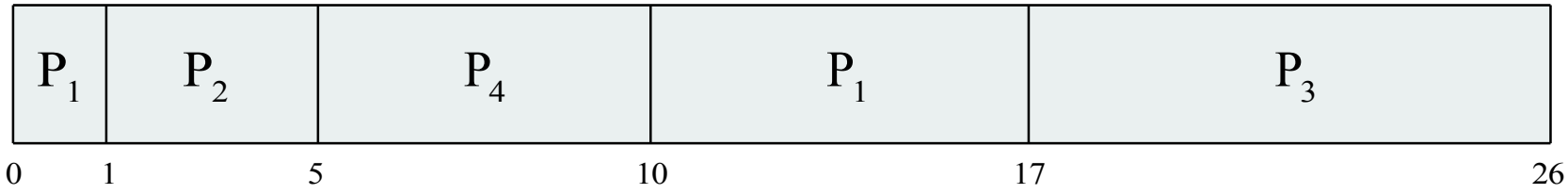
- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7
- Turnaround time: P1 = 9; P2 = 24; P3 = 16; P4 = 3

Review

- Process/Thread Scheduling
  - Shortest Remaining Time Next
    - Preemptive; varying arrival time

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- Gantt Chart:



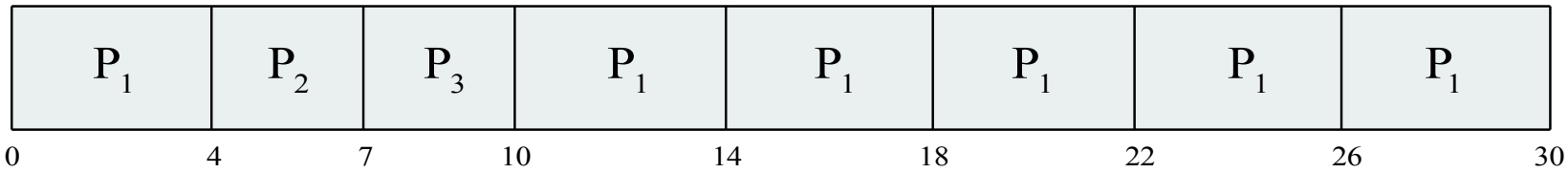
- Average waiting time = 6.5

Review

- Process/Thread Scheduling
  - Round Robin (RR)
    - Each process is assigned a time interval (quantum) during which it is allowed to run

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Assume quantum q = 4. The Gantt chart is:



- q should be large compared to context switch time
  - E.g., q usually 10ms to 100ms, context switch < 10 usec

Review

- Process/Thread Scheduling
  - Priority Scheduling
    - schedule the process with the highest priority

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Assume non-preemptive, all processes are ready at time 0
- Gantt Chart:

- Average waiting time = (1+6+16+18)/5 = 8.2

Review

- Inter-Process Communications (IPC)
  - Two basic methods
    - Shared memory
    - Message passing
  - Race Condition (Data Race)
    - two or more processes are reading or writing some **shared data** and the final result depends on who runs precisely when
  - Critical Region
    - A piece of code that **accesses a shared variable** and must not be concurrently executed by more than one thread
      - Multiple threads executing critical section can result in a race condition.
      - Need to support **mutual exclusion**

Review

- Example of race condition
  - Two threads perform “counter = counter +1”
    - “counter” is a shared variable; initially counter = 50

OS	Thread1	Thread2	(after instruction) PC	%eax	counter
			100	0	50
	mov 0x8049a1c, %eax		105	50	50
	add $0x1, %eax		108	51	50
interrupt					
save T1’s state					
restore T2’s state			100	0	50
		mov 0x8049a1c, %eax	105	50	50
		add $0x1, %eax	108	51	50
		mov %eax, 0x8049a1c	113	51	51
interrupt					
save T2’s state					
restore T1’s state			108	51	50
	mov %eax, 0x8049a1c		113	51	**51**

Review

- Solutions of mutual exclusion
  - Software solution
    - Disabling interrupts
      - Single processor only
      - Use for kernel
    - Strict alternation
      - Strict ordering
      - Busy waiting
    - Peterson's solution
      - Busy waiting
  - Hardware solution
    - TSL/XCHG
      - Work on multiprocessors
      - Busy waiting

Review

- Solutions of mutual exclusion
  - Pthread locks

```
pthread_mutex_t lock;
...
pthread_mutex_lock(&lock);
x = x + 1; // or whatever your critical section is
pthread_mutex_unlock(&lock);
```

Review

- Semaphore
  - Semaphore **S** – integer variable
    - Can only be accessed via two indivisible (atomic) operations
      - **down()** and **up()**
        - originally called **P()** and **V()**
        - also called **wait()** and **signal()**
    - meaning of **down()** operation (atomic):

```
down(S) {
    while (S <= 0)
        ; // busy waiting
    S--;
}
```

    - meaning of **up()** operation (atomic):

```
up(S) {
    S++;
}
```

Review

- Semaphore Example
  - Readers-Writers Problem

READER:

```
While (1) {
      down(protector);
      rc++;
      if (rc == 1)  //first reader
            down(database);
      up(protector);

      read();

      down(protector);
      rc--;
      If (rc == 0) then // last one
            up(database);
      up(protector);
      ….
}
```

WRITER:

```
While (1) {
      generate_data();
      down(database);
      write();
      up(database);
}
```

Two semaphores:

database

protector

Initial: protector=1, database =1

rc =0

Review

- Conditional Variable
  - Allows a thread to wait till a condition is satisfied
  - Example:

```
int    thread1_done = 0;

pthread_cond_t    cv;
pthread_mutex_t  mutex;
```

Thread 1:

```
printf("hello ");
pthread_mutex_lock(&mutex);
thread1_done = 1;
pthread_cond_signal(&cv);
pthread_mutex_unlock(&mutex);
```

Thread 2:

```
pthread_mutex_lock(&mutex);
while (thread1_done == 0) {
      pthread_cond_wait(&cv, &mutex);
}
printf(" world\n");
pthread_mutex_unlock(&mutex);
```

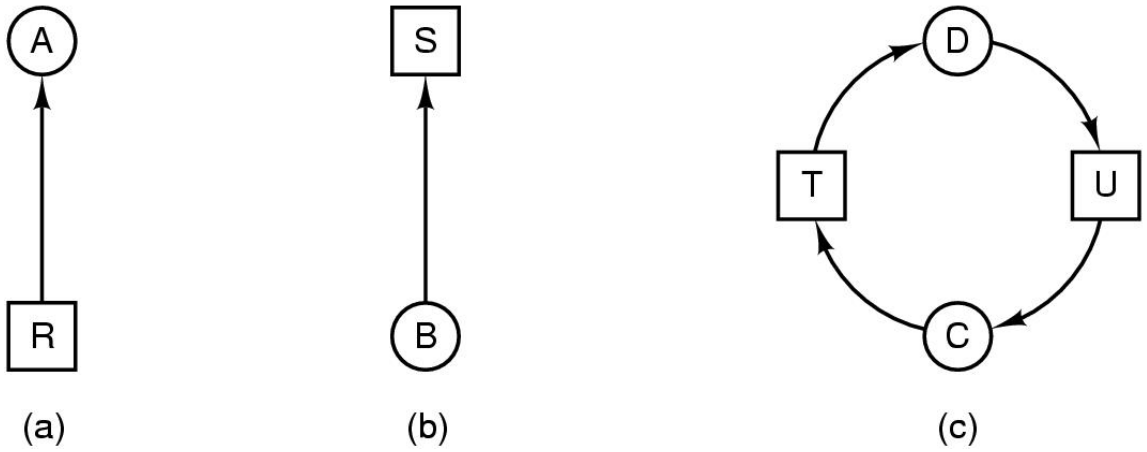
Review

- Deadlocks
  - Four conditions need to hold for a deadlock to occur

Condition	Description
Mutual Exclusion	Threads claim exclusive control of resources that they require.
Hold-and-wait	Threads hold resources allocated to them while waiting for additional resources
No preemption	Resources cannot be forcibly removed from threads that are holding them.
Circular wait	There exists a circular chain of threads such that each thread holds one more resources that are being requested by the next thread in the chain

Review

- Deadlocks
  - **Resource allocation graph**
    - resource R assigned to process A
    - process B is requesting/waiting for resource S
    - process C and D are in deadlock over resources T and U



Agenda

- ~~Midterm 1 Review~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.