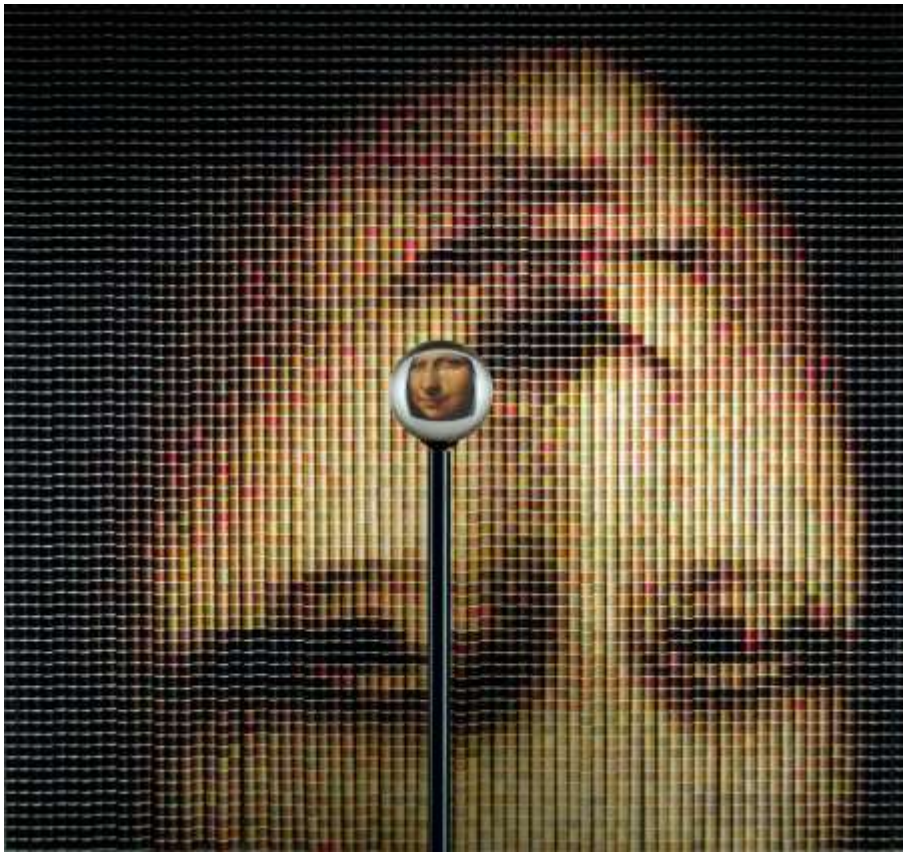
Creating Threads

COMP755



"I pledged California to a Northern Republic and to a flag that should have no treacherous threads of cotton in its warp, and the audience came down in thunder."

Thomas Starr King

Programming Assignment

- A simple programming assignment has been posted on Blackboard under assignments
- You can write the programs in C++, Java or another OO programming language
- Due by midnight on Friday, August 30

Short Term Schedule

	Wednesday, August 21 Introduction read chapter 1
Monday, August 26 OS structure read chapter 2	Wednesday, August 28 Concurrent Programming read sections 6.1 – 6.7
Monday, September 2 *Labor Day Holiday* *(no class)*	Wednesday, September 4 Concurrent Programming
Monday, September 9 Concurrent Programming	Wednesday, September 11 Deadlock & thread implementation read chapter 7
Monday, September 16 CPU scheduling & review read chapter 5	Wednesday, September 18 Exam 1
Monday, September 23 Queuing theory	Wednesday, September 25 Queuing theory
Monday, September 30 Performance prediction	Wednesday, October 2 Performance prediction
Monday, October 7 *Fall Break* *(no class)*	Wednesday, October 9 Memory management Chapter 8
Monday, October 14 Virtual memory Chapter 8	Wednesday, October 16 Virtual machines
Monday, October 21 Exam 2	

Reading Quiz

- Read sections 6.1 – 6.7 of the textbook and answer the quiz
- The quiz will be on Blackboard *(soon)*
- You get only one chance to take the quiz

Goals

- Understand the difference between processes and threads
- Review how the OS creates threads and processes
- See how to create multiple threads in
  - Java
  - Posix threads (pthreads)
  - Microsoft .Net

The Process

- All multiprogramming OSs are built around the concept of processes.
- A process is something akin to a program.
- User programs are processes.
- Some programs have multiple processes and the OS has several processes.
- A process has:
  - memory
  - ability to use the CPU (a thread)
  - resources allocated to it

Multiple Processes

- There are usually many processes existing in the system.
- If there is only one CPU, then only one thread can be running at any one time.
- Other threads might be waiting to run.
- Some (most) processes will be blocked waiting for something to happen (i.e. I/O)

Threads

- A process is a unit of resource ownership
- A thread is a unit of dispatching.
- A thread is another flow of control through the same program.
- Threads have an entry in the scheduling queue.
- Each thread has its own stack.

Creating Threads

- Each thread has its own set of registers and its own stack
- Threads share the program memory and resources (e.g. files) of their process.
- It is easier for the OS to create a thread than a process because it does not have to copy the memory image.

Comparison of Threads and Processes

- Cheaper to create a new thread
- Easier to switch between threads in the same process than between different processes
- Threads can easily share data
- Processes are protected from one another.

Threads and Processes

- A thread is a flow of execution in a process.
- Each thread belongs to one process.
- A process may have one or many threads.

If a thread opens a file, other threads in the program can

1. only read the file
2. access the file just as the opening thread
3. not access the file
4. access if they are children threads

Sharing the System

- Most programs perform a lot of I/O and therefore spend a lot of time waiting for I/O to complete.
- When a program is waiting for I/O, another program can be running.
- Some programs wait for a very long time for input (such as input from a keyboard, mouse or network)

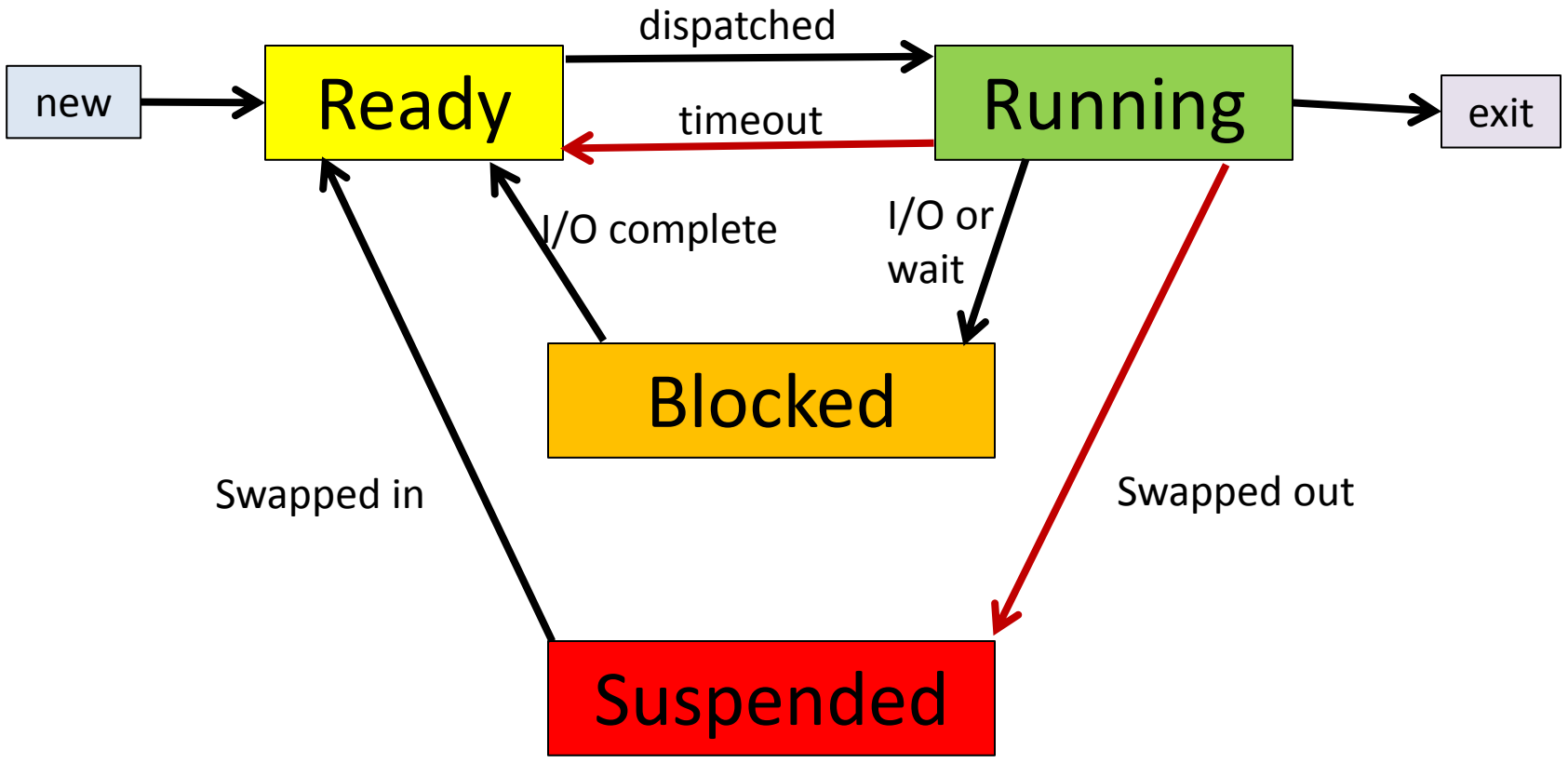
Process Creation

- When does a process get created?
  - When a program is started
  - Created by OS to provide a service to a user (ex: printing a file)
  - Spawned by an existing process
- When a process is created, the OS
  - Builds the data structures that are used to manage the process
  - Allocates address space in main memory to the process

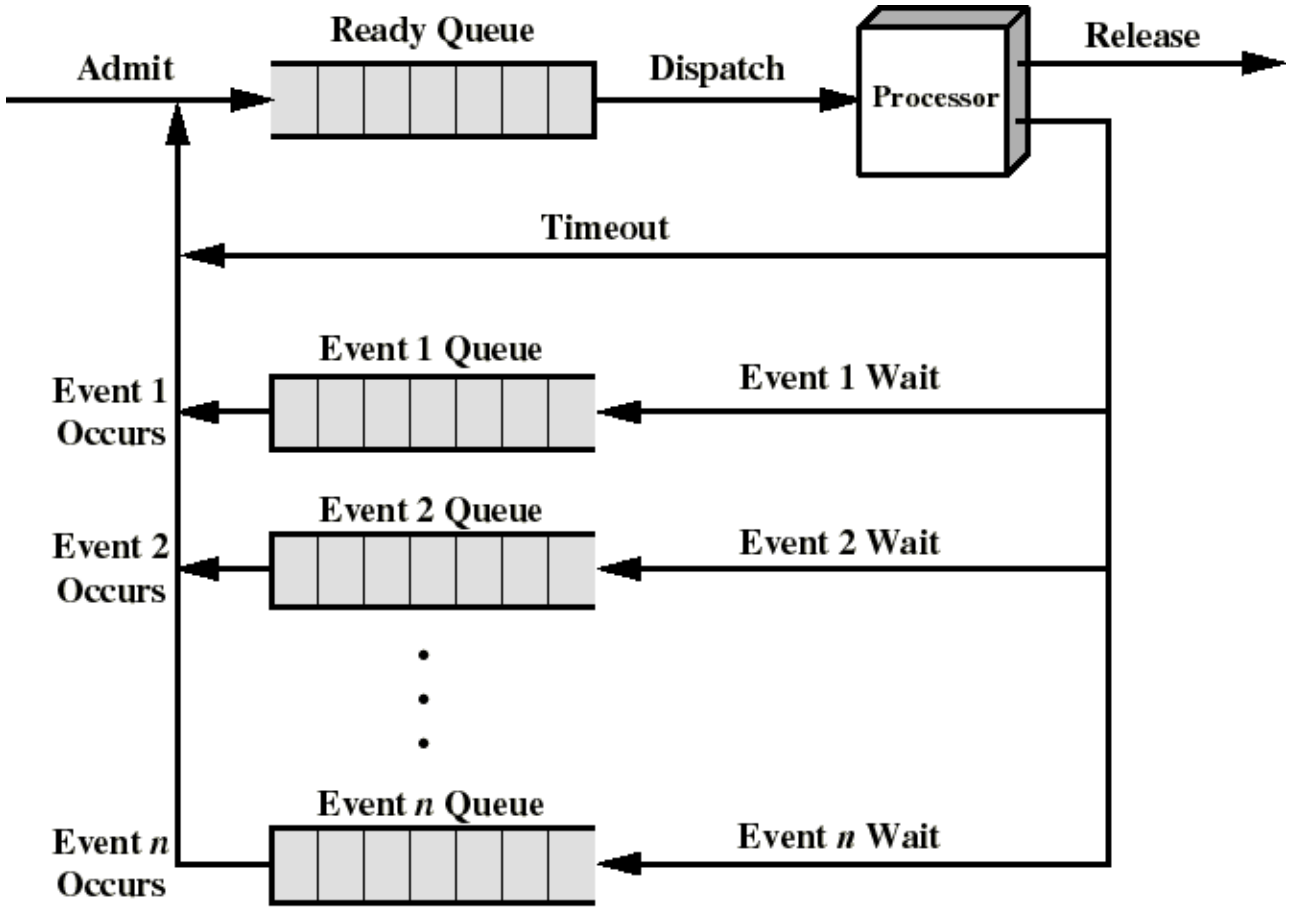
Process Termination

- User quits an application (e.g. word processor)
- Task completes
- Error and fault conditions

Process States



Queuing Model View of Processes



Process States

- **Running**: The process is being executed
- **Ready**: The process is prepared to execute when given the opportunity
- **Blocked**: The process can not execute until some event occurs, such as the completion of an I/O operation. There are many reasons a process can be blocked.

Process Transitions

- **Ready → Running:** When the CPU is available, the dispatcher selects a new process to run
- **Running → Ready:** The running process has run long enough for now; the running process gets interrupted because a higher priority process is in the ready state (preempted)

Process Transitions (cont.)

- **Running → Blocked**: When a process requests something for which it must wait (e.g., initiates I/O, wait for input from another process)
- **Blocked → Ready**: When the event for which it was waiting occurs
- **Running → Exit**: A process is done or has failed.

Dispatcher

- A part of the OS that gives the processor from one process to another
- Selects a process from the queue to execute after interrupt or process termination
- Prevents a single process from monopolizing the processor time

The dispatcher decides what state transition?

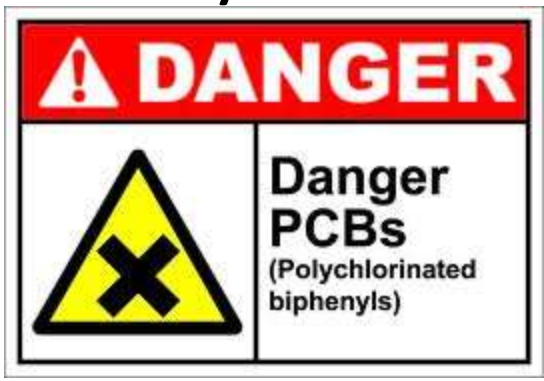
1. Running to Blocked
2. Ready to Running
3. Running to Ready
4. New to Ready

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Process Control Structures

Process control block (PCB) is the collection of attributes of the process that are used by OS for process control



Thread control block (TCB) contains information about a thread.

TCB Contents

- CPU State Information
  - What the CPU needs to run this thread
  - User-visible registers
  - Program Counter
  - Processor State Word (PSW)
- Scheduling and state information
  - Process state
  - priority
  - event for which the thread is waiting (if blocked)

PCB Contents

- Process Identification Information
  - Processor Identifier (PID)
    - 16 bit integer unique for each process
  - User identifier who is responsible for the job
- Memory management
  - pointers to segment/page tables assigned to this process
- Resource ownership and utilization
  - resources in use: opened files, I/O devices
- Process privileges

Process Creation

- Assign a unique Processor ID (PID).
- Allocate RAM for the PCB and TCB
- Initialize the PCB and TCB
- Allocate RAM for the program (code, data, stack and heap)
- Copy the program into the RAM
- Put the PCB on the list of PCBs
- Put TCB on ready list

Creating a new Process

- In C or C++ under Unix, you can create a new process with the fork function

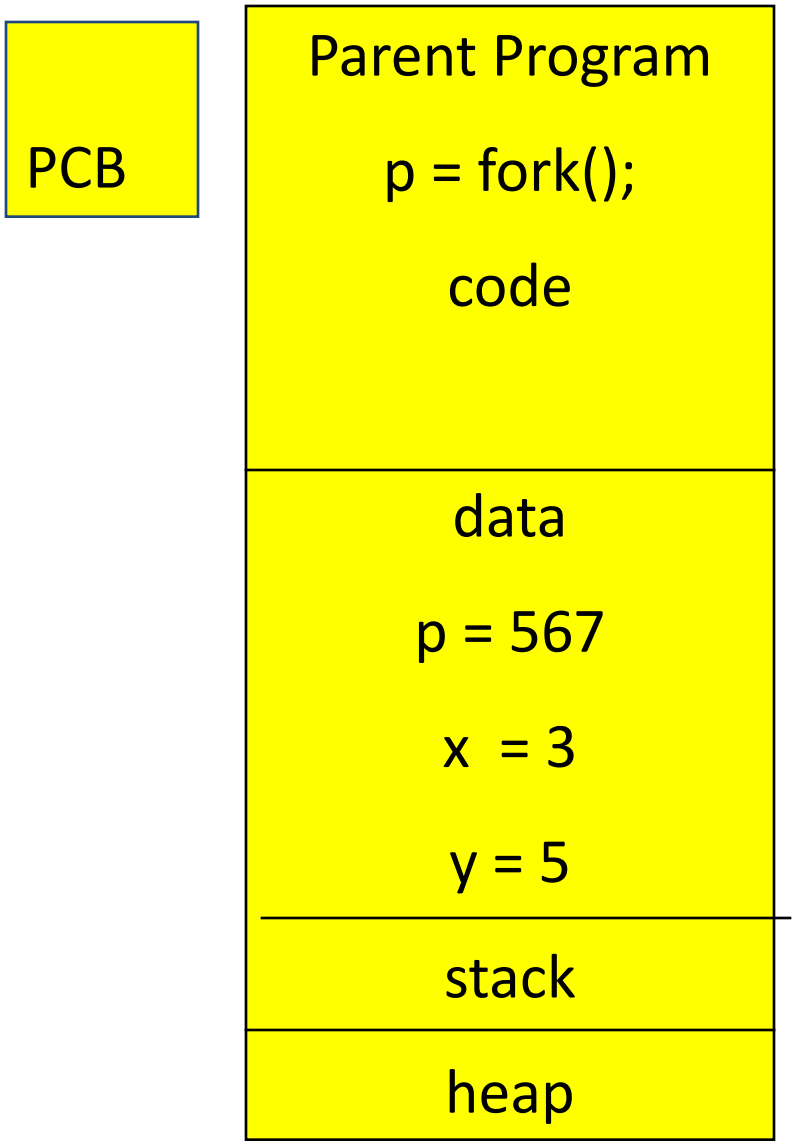
```
int fork();
```

- When fork is called, the RAM of the program is copied to another location is RAM. A new process is created to run in the new program address space. The new process is put on the ready list.

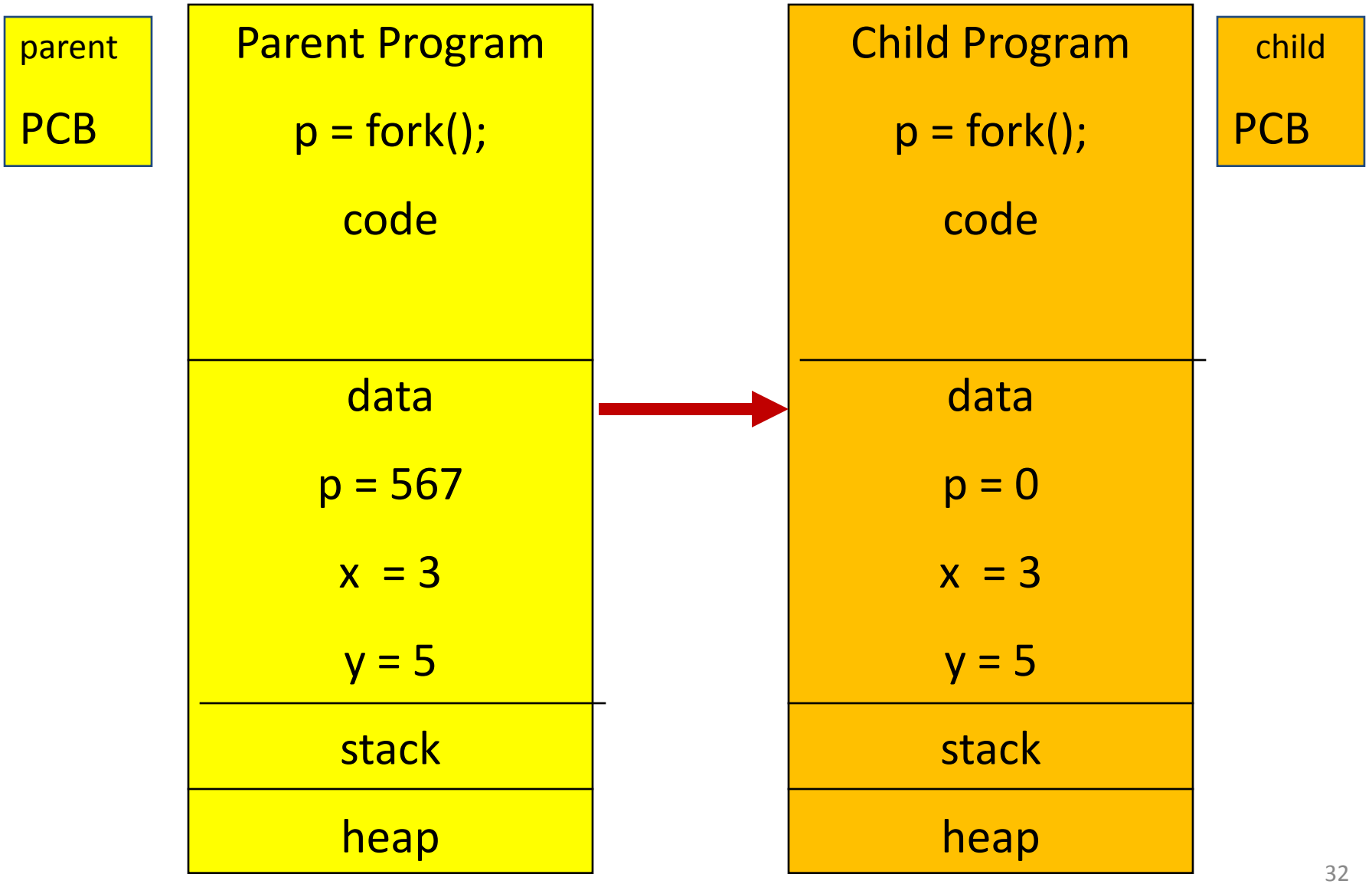
fork function

- The return value of the fork function is:
- -1 = Error
- 0 = This is the process that was just created.
- number = This process is the parent that just executed the fork function. The number returned is the PID of the child process.

Fork Action



Fork Action


parent
PCB
Parent Program
p = fork();
code
data
p = 567
x = 3
y = 5
stack
heap
Child Program
p = fork();
code
data
p = 0
x = 3
y = 5
stack
heap
child
PCB

```
/*  Example fork program */

int     counter = 0;     /* loop counter */
int     pid;             /* processor ID of child */
int     waitinc;         /* wait time in seconds */
char    id;              /* display identifier */

pid = fork();
if (pid < 0) { cout << "Fork error"; exit(8);}
else if (pid == 0) {   // if this is the child process
     id = 'C';
     waitinc = 1;
} else {                          // this is the parent process
     id = 'P';
     waitinc = 2;
}
for (i = 0; i < 3; i++) {
     counter++;
     cout << id << counter << " ";
     sleep(waitinc);
}
```

What is the likely output from the previous fork program

1. P1 P2 P3 C4 C5 C6
2. P1 P2 P3 C1 C2 C3
3. P1 C1 C2 P2 C3 P3
4. P1 C2 P3 C4 C5 P6

Starting Another Program

- Application programs are processes in the Windows world.
- You can start a new process with the **`_spawn`** functions.
- The process calling **`_spawn`** can wait for the child to complete, continue running or be overlaid.

Versions of Spawn

- There are many versions of spawn
- _spawnl _spawnv, _spawnle, _spawnve, _spawnlp, _spawnvp, _spawnlpe, _spawnvpe

suffix	description
e	pass environmental variable
p	use PATH environmental variable to find file to execute
l	command line arguments passed as separate variables
v	argv array contains command line arguments

Unix Program Execution

- The exec functions will start execution of a specified program.
- The program will overlay the existing program.
- Sequential execution

#include <unistd.h>
int execl(const char *_path_, const char *_arg0_, ..., const char *_argn_, char * /*NULL*/);

int execv(const char *_path_, char *const _argv_[ ]);

and several others

Sharing Memory

- Creating a new process makes a completely new address space with a copy of the original program.
- None of the variables in the parent copy are accessible by the child process.
- Programmers can make a shared memory segment to communicate.
- Multiple threads can share the same process memory.

POSIX Threads

- The POSIX operating system standard defines a thread library that is portable to any system supporting the POSIX standard.
- The **`pthread_create`** function creates a new thread.
- You can learn about POSIX threads from **`man pthread`**

Pthread Programming in C

```
#include <pthread.h>
 int pthread_create(
      pthread_t            *thread,
      const pthread_attr_t  *attr,
      void *               (*start_routine)(void *),
      void                  *arg);
```

compile as

```
cc myprog.c -lpthread
```

Pthread Example

```
#include <pthread.h>
pthread_t   threadObj;
int     aNumber = 47;

 pthread_create(&threadObj, NULL, myfunc,
                    &aNumber);

void * myfunc(void *arg) { int x = *(int*)arg; }
```

Pthread Example *(cont.)*

- The thread object of type **`pthread_t`** is used by the **`pthread`** system to store thread information. No initialization of this object is necessary.
- The **`pthread_create`** function calls the **`myfunc`** function with a pointer to **`aNumber`** as the function argument.
- **`myfunc`** executes in parallel with the calling function.

Thread Termination

- Thread creation calls a function or method that will execute in parallel with the calling thread.
- The new thread will terminate when it would return to the calling function.
- Threads can also be terminated by:

```
void  pthread_exit(void value);
```

Thread Creation with Visual Studio .NET

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES    lpThreadAttributes, // SD
  SIZE_T       dwStackSize,            // initial stack size
  LPTHREAD_START_ROUTINE lpStartAddress,    // function
  LPVOID       lpParameter,            // thread argument
  DWORD        dwCreationFlags,        // creation option
  LPDWORD  lpThreadId                  // thread identifier
);
```

.NET parameters

- ThreadAttributes - security descriptor for the new thread or null
- dwStackSize – stack size or zero for default
- lpParameter - single parameter value passed to the thread
- dwCreationFlags - CREATE_SUSPENDED or 0 to start now
- lpThreadId – [out] returned thread ID or null

Started Thread Function

```
DWORD WINAPI yourfunction(
  LPVOID lpParameter   // thread data
);
```

The return value indicates the thread's success or failure.

The parameter is the value passed to CreateThread

Threads in Java

- Threads are built into Java as part of the standard class library.
- There are two ways to create threads in Java:
  - Extend the class **`Thread`**
  - Implement the interface **`Runnable`**

run Method

- Both means of creating threads in Java require the programmer to implement the method:

```
public void run() { … }
```

- When a new thread is created, the **run** method is executed in parallel with the calling thread.

Extending Java Thread Class

```
class Example extends Thread {
  int classData;   // example data
  // optional constructor
  Example(int something) {
     classData= something;
  }

  public void run() {
  // runs in parallel
  . . .
  }
}
```

Starting a Thread Object

- Parallel execution of the run method of a Thread object is initiated by:

```
// create Thread Object
Example xyz = new Example(143);
// start execution of run method
xyz.start();
```

Runnable Interface

- Java does not support multiple inheritance. If a class extends Thread, it cannot extend another class.
- Programmers frequently want to use multiple threads and extend another class, such as Applet.
- The Runnable interface allows a program use multiple threads and inheritance.

Implementing Runnable Interface

```
class RExample implements Runnable {
  int classData;   // example data
  // optional constructor
  RExample(int something) {
     classData= something;
  }

  public void run() {
  // runs in parallel
  . . .
  }
}
```

Starting a Runnable Object

- Parallel execution of the run method of a Runnable object is initiated by:

```
// create Thread Object
RExample xyz = new RExample(143);
// start execution of run method
new Thread(xyz).start();
```

Java Thread Termination

- Similar to pthreads, Java threads terminate when the run method returns
- The System.exit(int) method will terminate the entire process

A difference between pthreads and java is

1. Java starts processes while pthreads creates threads
2. pthreads allows you to specify the method to run in parallel, not java
3. pthreads runs in true parallelism, not java
4. All of the above

Only One Run

- An annoying feature about Java threads is that they always start a method named run. Other thread systems allow you to specify the method to be executed.
- The run method has no parameters. Data must be passed to the method through common variables.
- A thread can only be started once.

```
public class MultiRun extends Thread {
    int counter = 0;
    public void run() {
        System.out.print(counter);
        counter++;
    }
    public static void main(String[] args) {
        MultiRun yarn = new MultiRun();
        yarn.start();
        yarn.start(); // This causes an error
    }
}
```

```
public class MultiRun extends Thread {
    static int counter = 0;
    public void run() {
        System.out.print(counter);
        counter++;
    }
    public static void main(String[] args) {
        MultiRun yarn = new MultiRun();
        yarn.start();
        MultiRun yarn2 = new MultiRun();
        yarn2.start();
    }
}
```

What Does This Display?

```
public class MultiRun extends Thread {
    static int counter = 0;
    public void run() {
        System.out.print(counter);
    }
    public static void main(String[] args) {
        MultiRun yarn = new MultiRun();
        yarn.start();
        counter++;
        MultiRun yarn2 = new MultiRun();
        yarn2.start();
    }
}
```

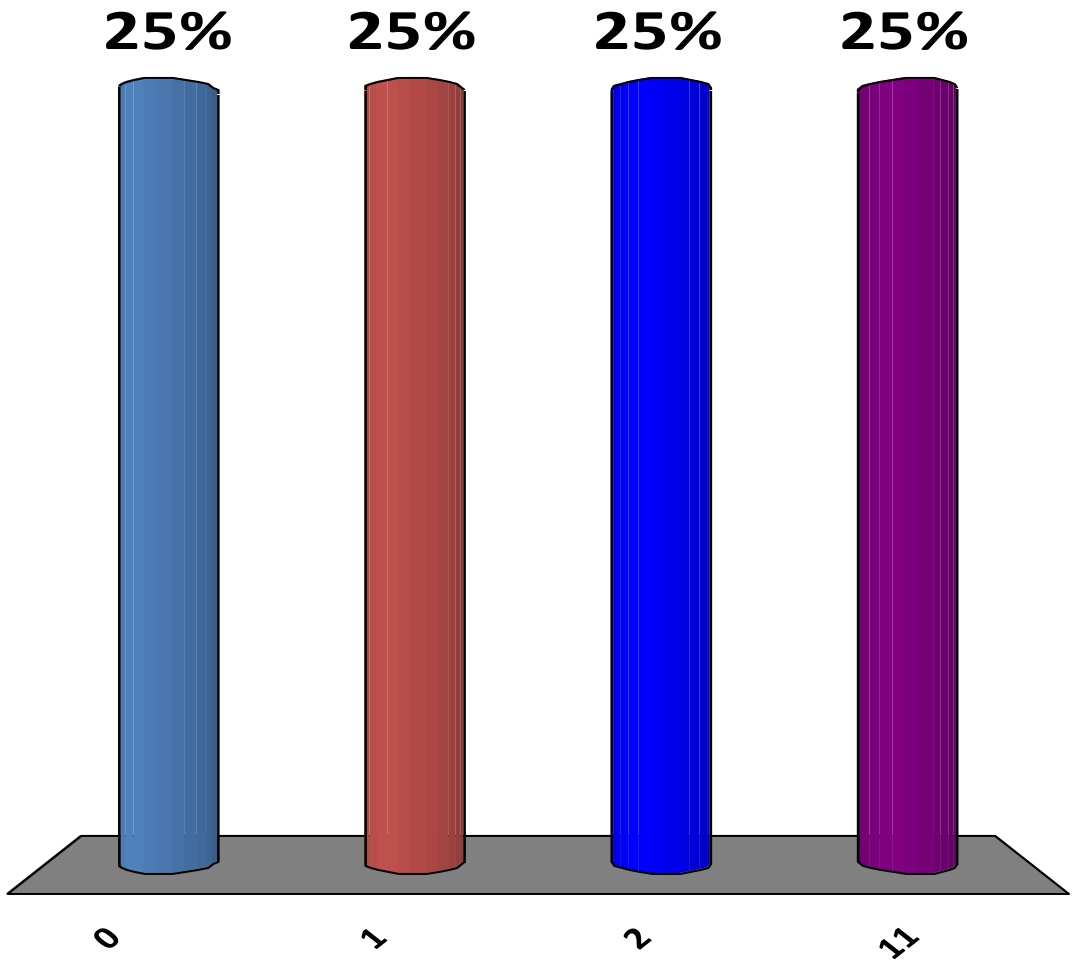
What Does MultiRun Display?

A. 00

B. 01

C. 02

D. 11



Programming Assignment

- A simple programming assignment has been posted on Blackboard under assignments
- You can write the programs in C++, Java or another OO programming language
- Due by midnight on Friday, August 30

Reading Quiz

- Read sections 6.1 – 6.7 of the textbook and answer the quiz
- The quiz will be on Blackboard *(soon)*
- You get only one chance to take the quiz