

CST 347 ?? Real Time OS

CST 347 – Real Time OS

Oregon TECH

Lecture 04 – Processes

Troy Scevers



Definitions

Definitions

- Concurrency
 - The appearance that threads are running simultaneously even though there is a single CPU.
- Context
 - The “processor” state of a block of executing code. This includes all registers required to uniquely identify this chain of execution.
- Process
 - A group of instructions along with the context defining the execution “state (s)” of those instructions.

Objectives

Objectives

- To introduce the notion of a process
 - A program in execution, which forms the basis of all computation
- To describe the various features of processes, including scheduling, creation and termination, and communication

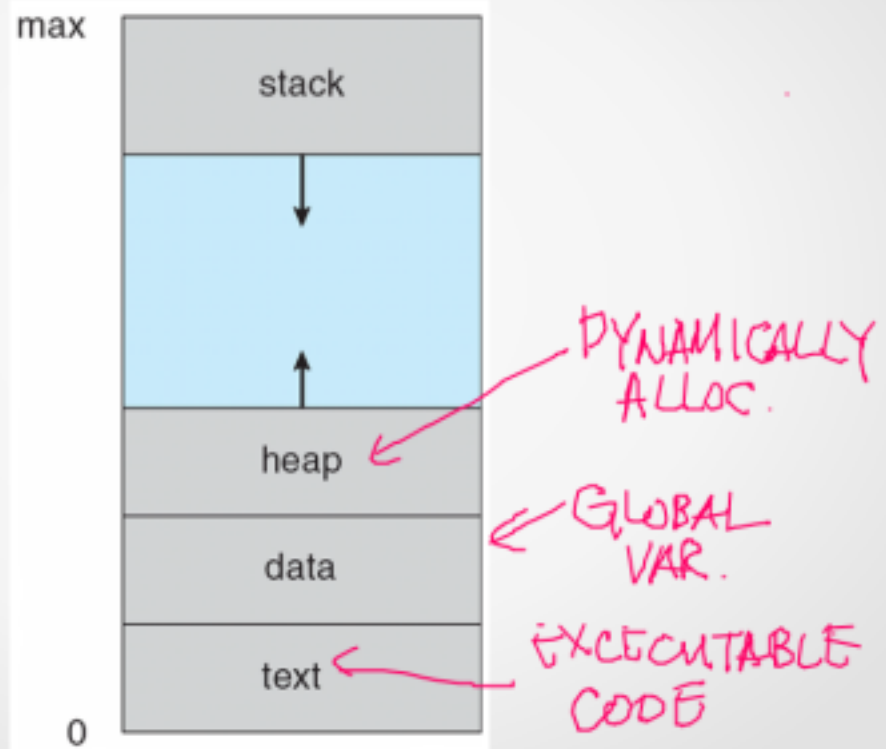
Process Concept

Process Concept

- An operating system executes a variety of programs:
 - Batch system – **jobs**
 - Time-shared systems – **user programs** or **tasks**
- **Process** – a program in execution; process execution must progress in sequential fashion
 - Multiple parts
 - The program code, also called **text section**
 - Current activity including **program counter**, processor registers
 - **Stack** containing temporary data
 - Function parameters, return addresses, local variables
 - **Data section** containing global variables
 - **Heap** containing memory dynamically allocated during run time
- Program is passive entity stored on disk (**executable file**), process is active
 - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
 - Consider multiple users executing the same program

Process in Memory

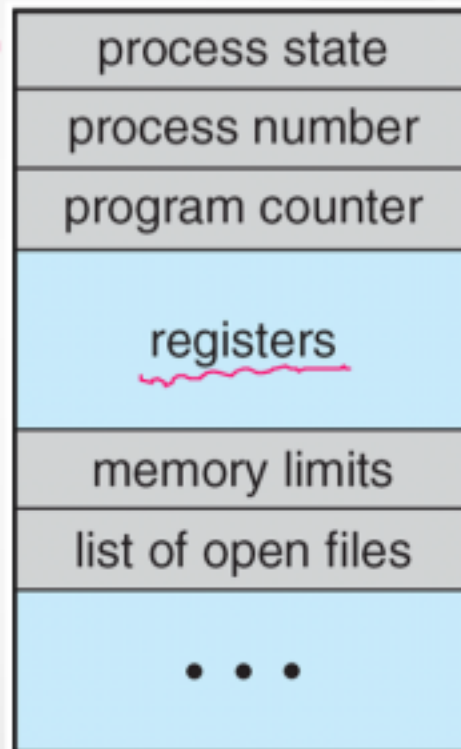
Process in Memory



Process Control Block (PCB)

Process Control Block (PCB)

- Information associated with each process (also called **task control block**)
 - Process state – running, waiting, etc
 - Program counter – location of instruction to next execute
 - CPU registers – contents of all process-centric registers
 - CPU scheduling information- priorities, scheduling queue pointers
 - Memory-management information – memory allocated to the process
 - Accounting information – CPU used, clock time elapsed since start, time limits
 - I/O status information – I/O devices allocated to process, list of open files



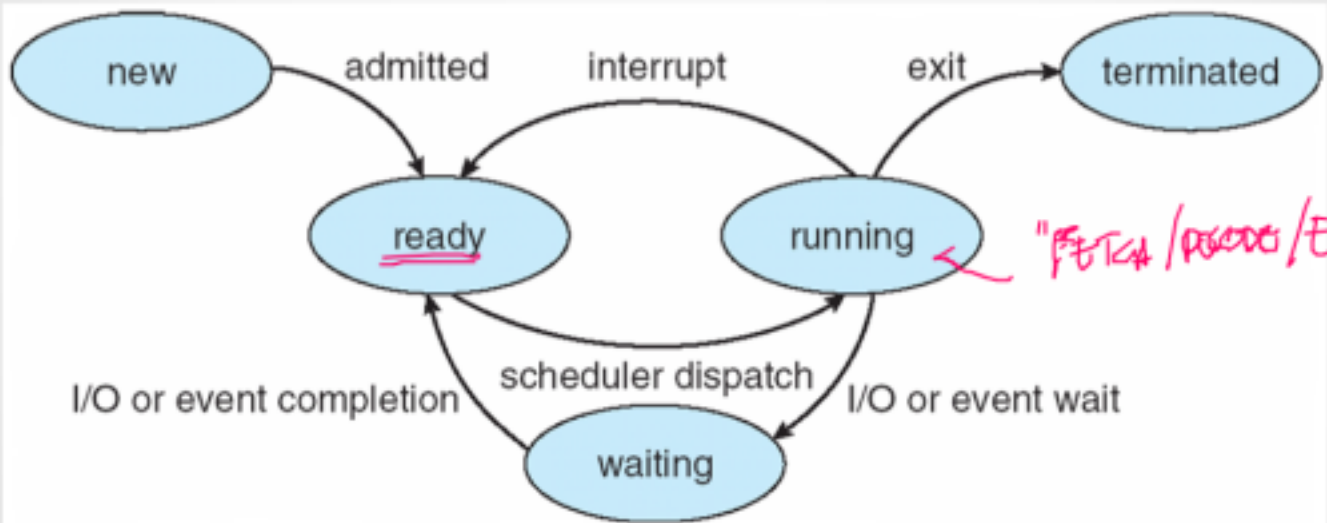
Process State

Process State

- As a process executes, it changes **state**
 - new: The process is being created
 - running: Instructions are being executed
 - waiting: The process is waiting for some event to occur ← "BLOCKED"
 - ready: The process is waiting to be assigned to a processor
 - terminated: The process has finished execution

Diagram of Process State

Diagram of Process State

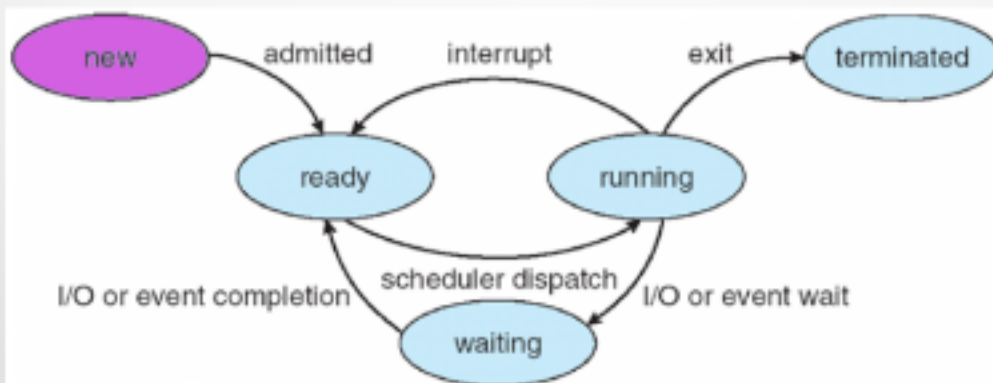


Process State Transition

Process State Transition

- Process Creation

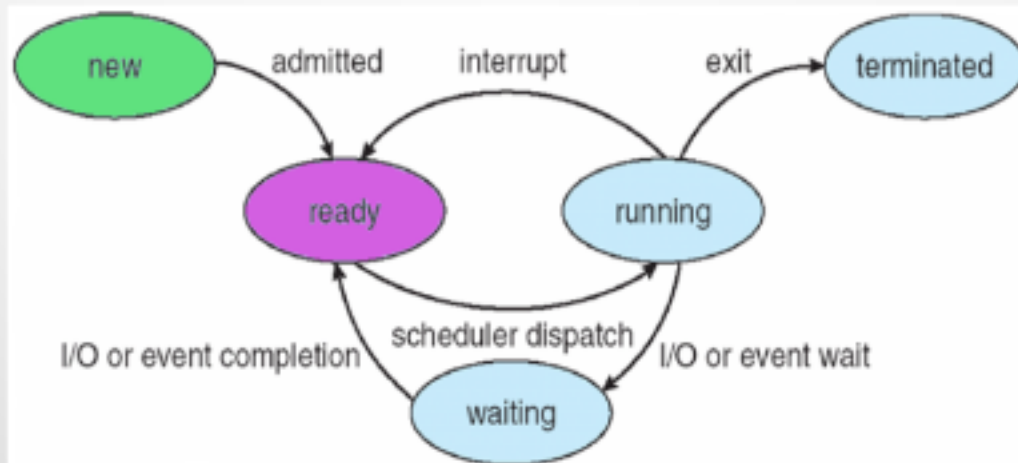
- Scheduler creates the Process Control Block and places it in the new list
 - Create process data segment.
 - Create process code segment.
 - Load op codes from disk into memory
 - Build run-time stack.



Process State Transition

Process State Transition

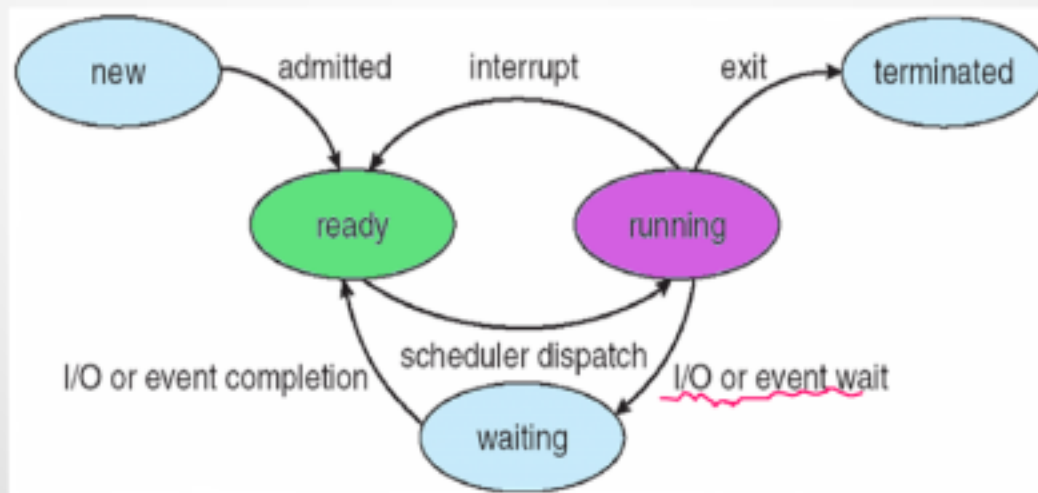
- Process is admitted by the system and goes to the ready state
 - Process system call is done and ready for execution
 - Process Control Block marked as active



Process State Transition

Process State Transition

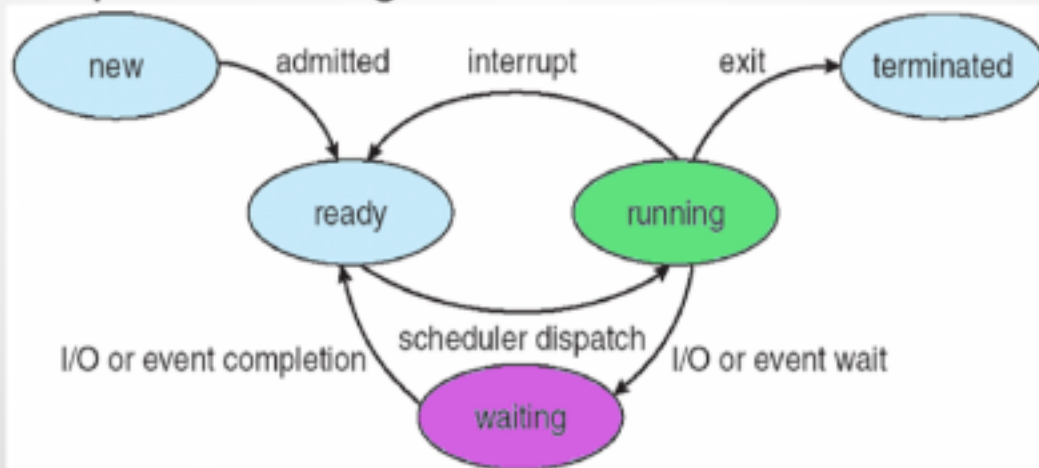
- Scheduler dispatch
 - Process Control Block is switched into the CPU based on Scheduling algorithm



Process State Transition

Process State Transition

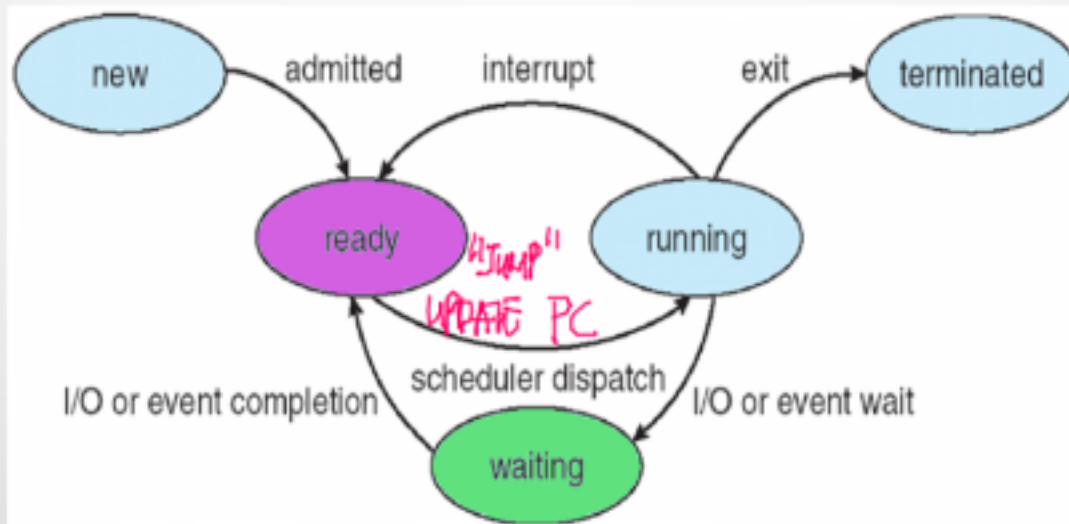
- IO or event Wait
 - Process requested some unavailable resource
 - Process Control Block is switched out of the CPU and put on waiting list



Process State Transition

Process State Transition

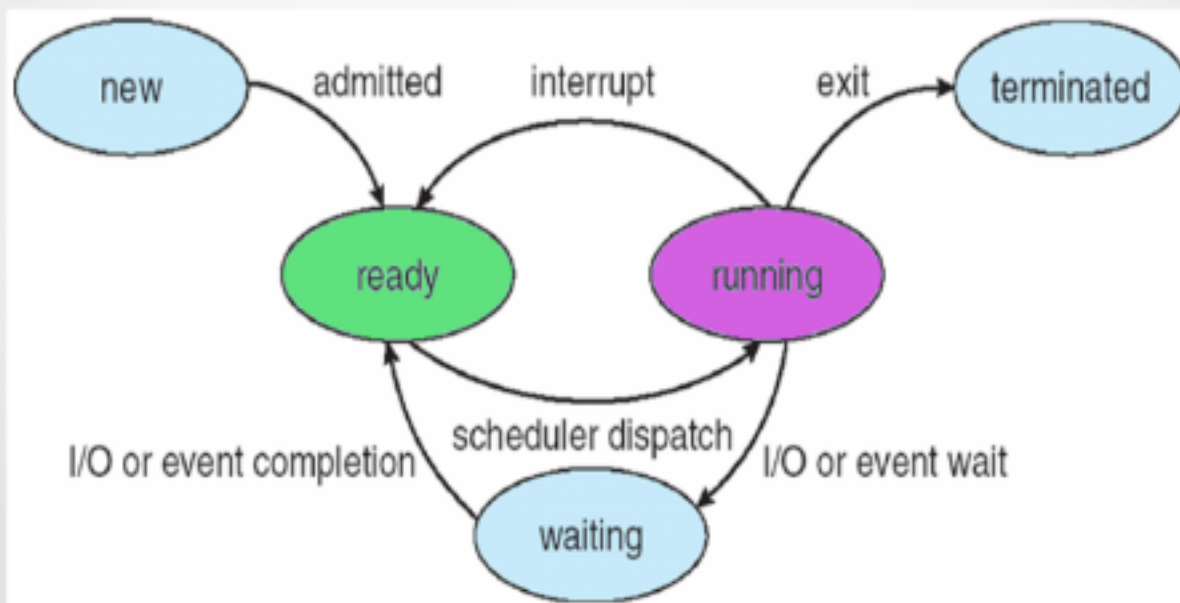
- Requested resource becomes available
 - Process Control blocked moved from waiting to ready (or active) state



Process State Transition

Process State Transition

- Process continues to run based on scheduling algorithm



Process State Transition

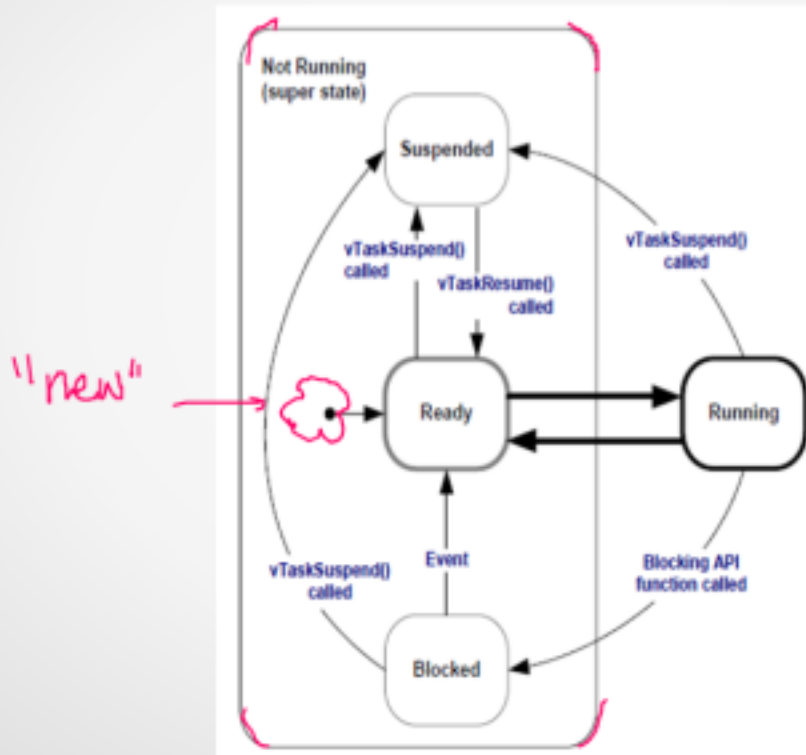
Process State Transition

- Process Completes or is terminated by another process
 - Process Resources are returned to OS
 - Process Memory Segment is cleaned up
 - Process Runtime stack is cleaned up



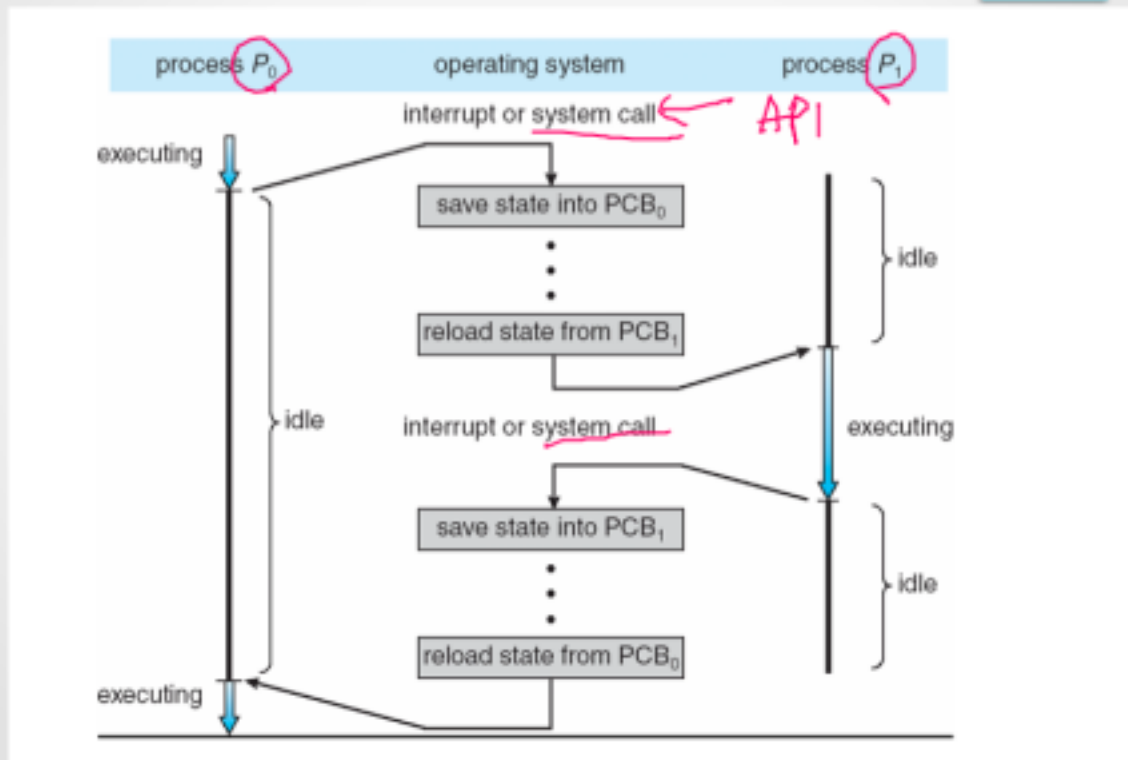
FreeRTOS Task State

FreeRTOS Task State



CPU Switch From Process to Process

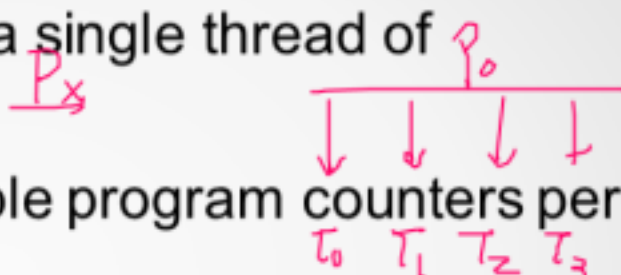
CPU Switch From Process to Process



Threads

Threads

- So far, A process has a single thread of execution
- Consider having multiple program counters per process
 - Multiple locations can execute at once
 - Multiple threads of control → threads ← "TASK"
- Must then have storage for thread details, multiple program counters in PCB
- We'll talk more about threads next lecture

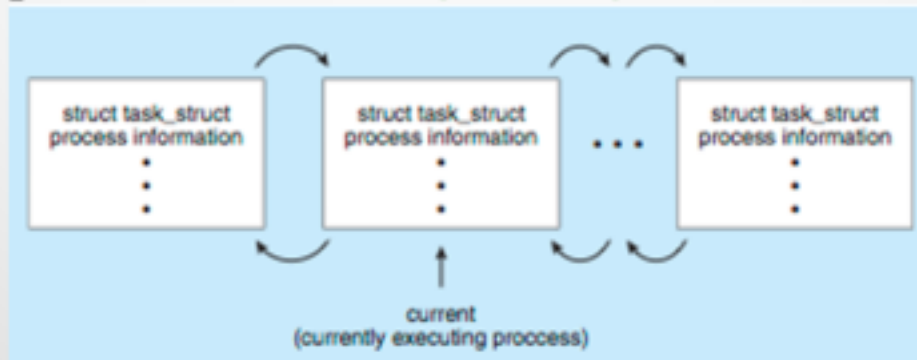


Process Representation in Linux

Process Representation in Linux

- Represented by the C structure

```
struct task_struct  
{  
    pid_t pid;           /* process identifier */  
    long state;          /* state of the process */  
    unsigned int time_slice; /* scheduling information */  
    struct task_struct *parent; /* this process's parent */  
    struct list_head children; /* this process's children */  
    struct files_struct *files; /* list of open files */  
    struct mm_struct *mm; /* address space of this process */  
}
```



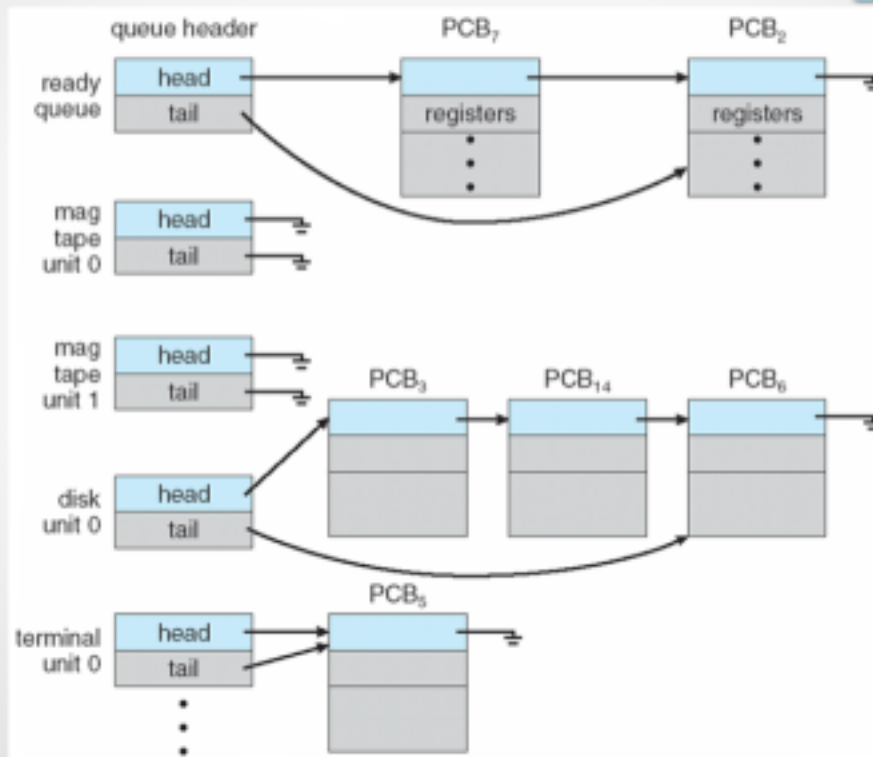
Process Scheduling

Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
 - **Job queue** – set of all processes in the system
 - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
 - **Device queues** – set of processes waiting for an I/O device
 - Processes migrate among the various queues

Ready Queue And Various I/O Device Queues

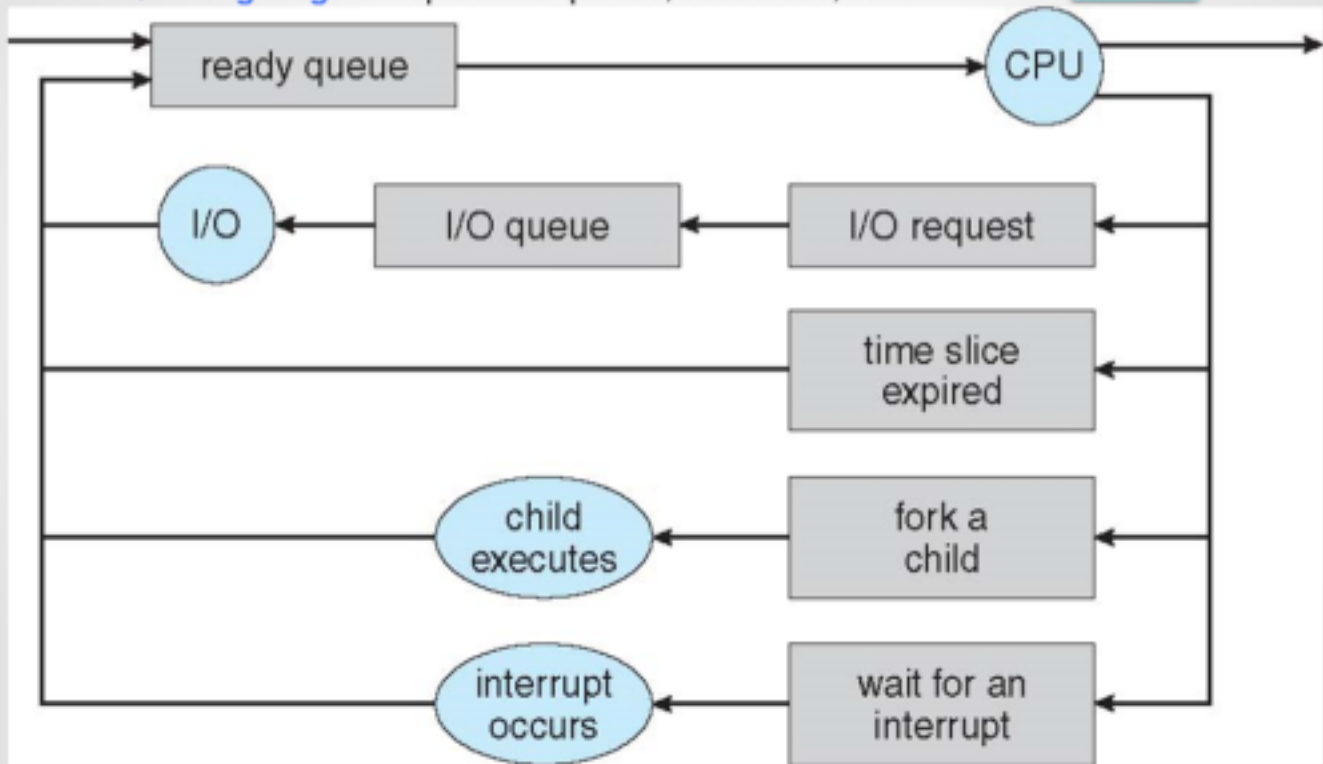
Ready Queue And Various I/O Device Queues



Representation of Process Scheduling

Representation of Process Scheduling

- **Queuing diagram** represents queues, resources, flows



Schedulers

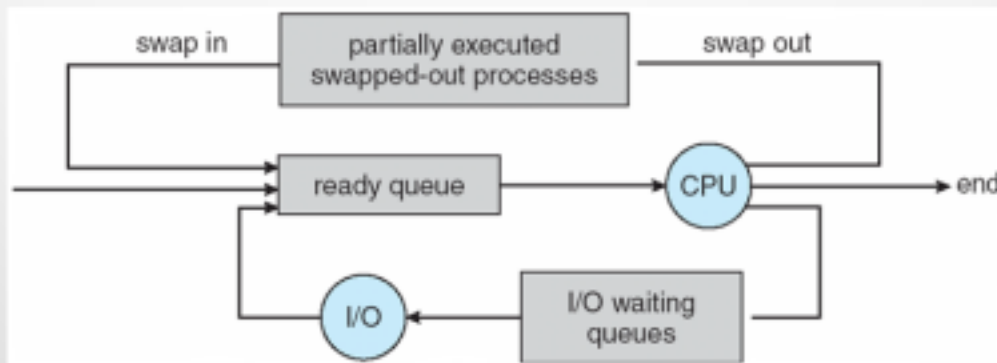
Schedulers

- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
 - Sometimes the only scheduler in a system
- Short-term scheduler is invoked very frequently (milliseconds) \Rightarrow (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) \Rightarrow (may be slow)
- The long-term scheduler controls the **degree of multiprogramming**
- Processes can be described as either:
 - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
 - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good *process mix*

Addition of Medium Term Scheduling

Addition of Medium Term Scheduling

- Medium-term scheduler can be added if degree of multiple programming needs to decrease
 - Remove process from memory, store on disk, bring back in from disk to continue execution: swapping



Multitasking in Mobile Systems

Multitasking in Mobile Systems

- Some systems / early systems allow only one process to run, others suspended
- Due to screen real estate, user interface limits iOS provides for a
 - Single **foreground** process- controlled via user interface
 - Multiple **background** processes- in memory, running, but not on the display, and with limits
 - Limits include single, short task, receiving notification of events, specific long-running tasks like audio playback
- Android runs foreground and background, with fewer limits
 - Background process uses a **service** to perform tasks
 - Service can keep running even if background process is suspended
 - Service has no user interface, small memory use

Context Switch

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 via a **context switch**
- **Context** of a process represented in the PCB *INCL. REGISTERS (PC, SP, GPR)*
- Context-switch time is overhead; the system does no useful work while switching
 - The more complex the OS and the PCB → longer the context switch
- Time dependent on hardware support
 - Some hardware provides multiple sets of registers per CPU → multiple contexts loaded at once

Operations on Processes

Operations on Processes

- System must provide mechanisms for process creation, termination, and so on as detailed next

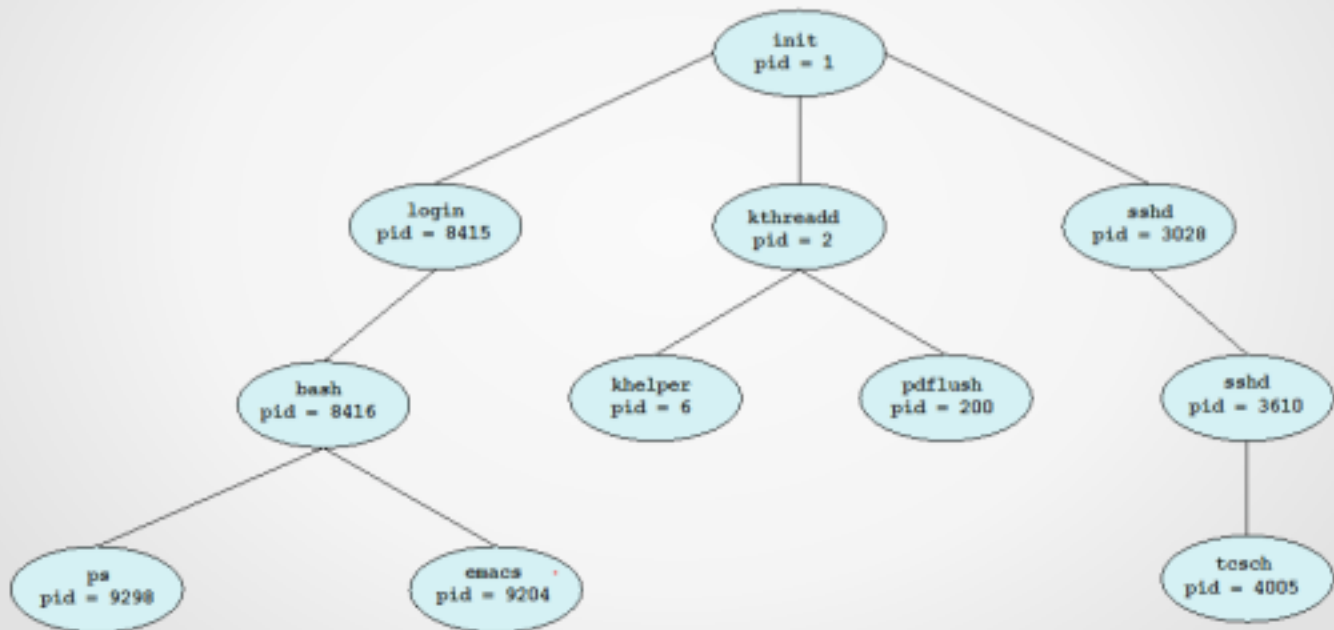
Process Creation

Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier (pid)**
- Resource sharing options
 - Parent and children share all resources
 - Children share subset of parent's resources
 - Parent and child share no resources
- Execution options
 - Parent and children execute concurrently
 - Parent waits until children terminate

A Tree of Processes in Linux

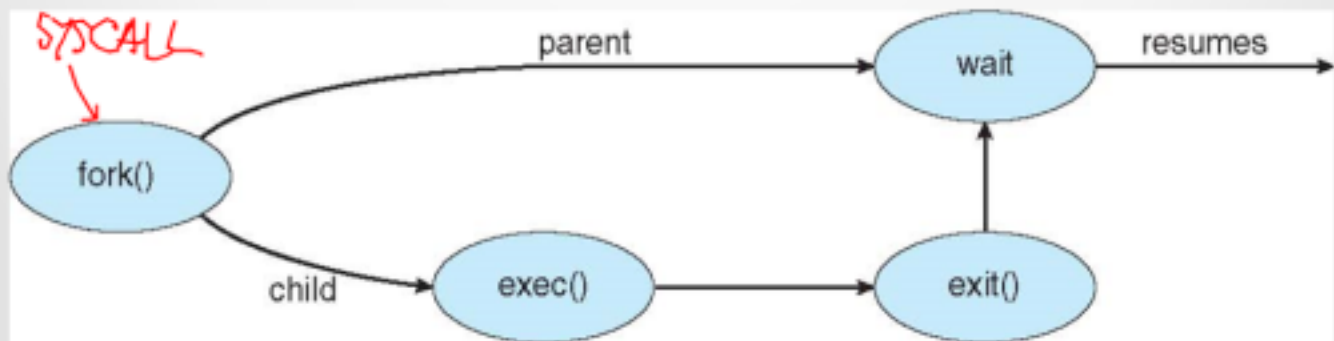
A Tree of Processes in Linux



Process Creation (Cont.)

Process Creation (Cont.)

- Address space
 - Child duplicate of parent
 - Child has a program loaded into it
- UNIX examples
 - **fork()** system call creates new process



C Program Forking Separate Process

C Program Forking Separate Process

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait(NULL);
        printf("Child Complete");
    }

    return 0;
}
```

PARENT EXEC (handwritten red text with arrows pointing to the parent process execution flow)

CHILD EXEC (handwritten red text with arrows pointing to the child process execution flow)

PARENT "BLOCKED" (handwritten red text with a circled 'X' and arrows pointing to the `wait(NULL);` line)

Creating a Separate Process via Windows API

Creating a Separate Process via Windows API

```
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    /* allocate memory */
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    /* create child process */
    if (!CreateProcess(NULL, /* use command line */
        "C:\\WINDOWS\\system32\\mspaint.exe", /* command */
        NULL, /* don't inherit process handle */
        NULL, /* don't inherit thread handle */
        FALSE, /* disable handle inheritance */
        0, /* no creation flags */
        NULL, /* use parent's environment block */
        NULL, /* use parent's existing directory */
        &si,
        &pi))
    {
        fprintf(stderr, "Create Process Failed");
        return -1;
    }
    /* parent will wait for the child to complete */
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    /* close handles */
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```


Process Termination

Process Termination

- Process executes last statement and asks the operating system to delete it (**exit()**)
 - Output data from child to parent (via **wait()**)
 - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort()**)
 - Child has exceeded allocated resources
 - Task assigned to child is no longer required
 - If parent is exiting
 - Some operating systems do not allow child to continue if its parent terminates
 - All children terminated - **cascading termination**

- Wait for termination, returning the pid:

```
pid_t pid; int status;
```

```
pid = wait(&status);
```

- If no parent waiting, then terminated process is a **zombie**
- If parent terminated, processes are **orphans**

Multiprocess Architecture ?? Chrome Browser

Multiprocess Architecture – Chrome Browser

- Many web browsers ran as single process (some still do)
 - If one web site causes trouble, entire browser can hang or crash
- Google Chrome Browser uses multiprocesses with 3 categories:
 - **Browser** process manages user interface, disk and network I/O
 - **Renderer** process renders web pages, deals with HTML, Javascript, new one for each website opened
 - Runs in **sandbox** restricting disk and network I/O, minimizing effect of security exploits
 - **Plug-in** process for each type of plug-in

