Processes

CSE 4001 Operating Systems Concepts

E. Ribeiro

January 26, 2022

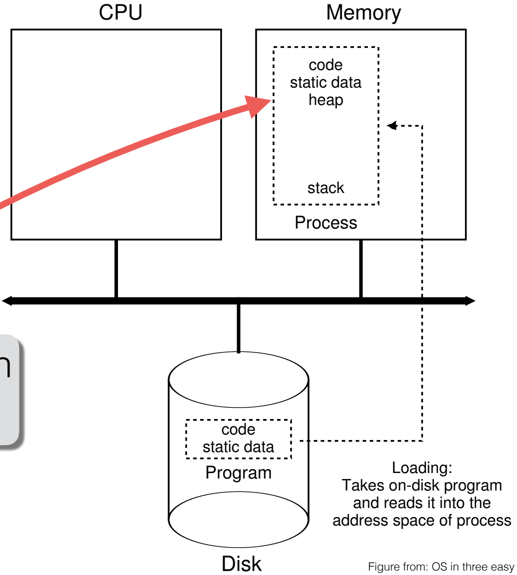
Outline

1. Processes

A process is a program in execution

What is a process?

A *process* is an abstraction of *a program in execution.*



Loading:
Takes on-disk program
and reads it into the
address space of process

Figure from: OS in three easy pieces

Main components of the address space

What is a process?

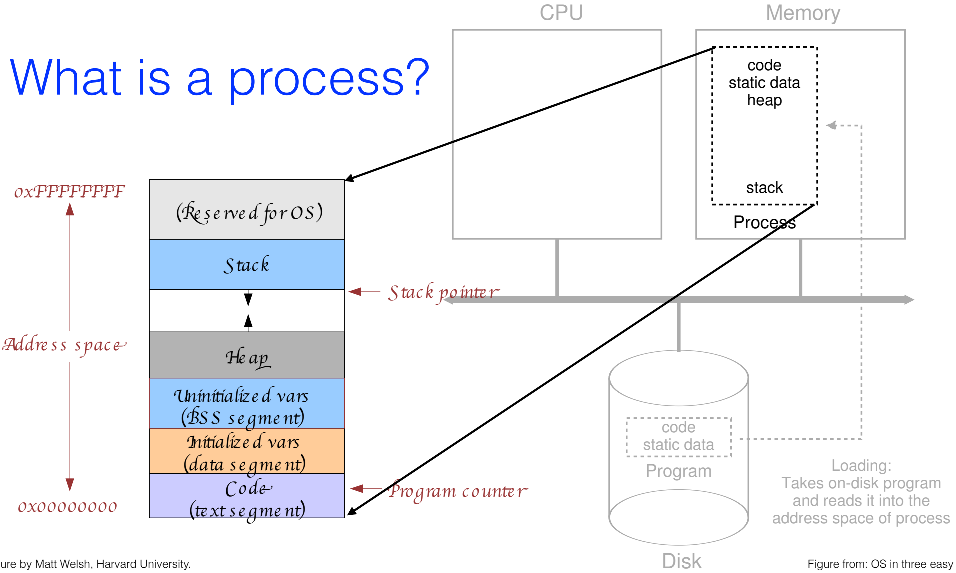


Figure by Matt Welsh, Harvard University.

Figure from: OS in three easy pieces

The code part of the address space

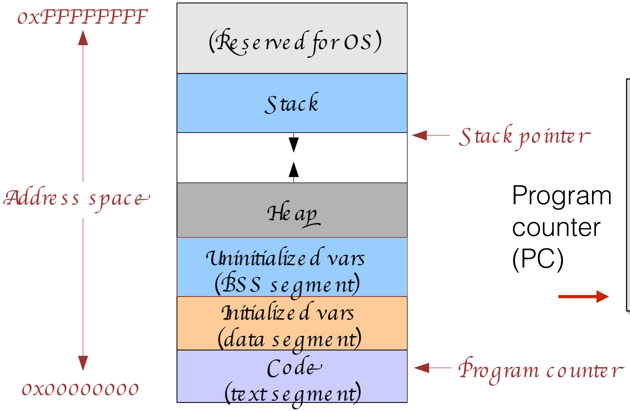


Figure by Matt Welsh, Harvard University.

Program counter (PC)

```
/* Hello World program */

#include<stdio.h>

main()
{
    printf("Hello World");

}
```

The OS view of a process

- Process state (ready, running, blocked, ...)
- The **address space** (how many possible addresses)
- The **code** of the running program
- The **data** of the running program
- An execution **stack** encapsulating the state of procedure calls
- The **program counter (PC)** indicating the address of the next instruction.
- A set of general-purpose **registers** with current values
- A set of operating system **resources**
  - open files, network connections, signals, etc.
- CPU scheduling info: process **priority**
- Each process is identified by its **process ID (PID)**

All these information is stored in a construct called **Process Control Block (PCB)**

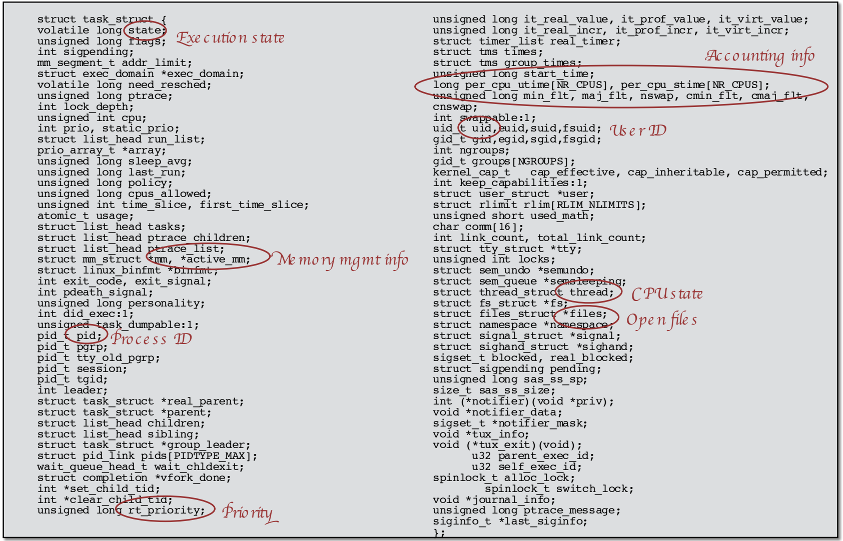
Adapted from original slide by Larry Zhang

The Process Control Block (PCB)

The OS maintains a PCB for each process. It is a data structure with many fields.

Defined in:

`/include/linux/sched.h`



```
struct task_struct {
volatile long state;
unsigned long flags;
int sigpending;
mm_segment_t addr_limit;
struct exec_domain *exec_domain;
volatile long need_resched;
unsigned long ptrace;
int lock_depth;
unsigned int cpu;
int prio, static_prio;
struct list_head run_list;
prio_array_t *array;
unsigned long sleep_avg;
unsigned long last_run;
unsigned long policy;
unsigned long cpus_allowed;
unsigned int time_slice, first_time_slice;
atomic_t usage;
struct list_head tasks;
struct list_head ptrace_children;
struct list_head ptrace_list;
struct mm_struct *mm, *active_mm;
struct linux_binfmt *binfmt;
int exit_code, exit_signal;
int pdeath_signal;
unsigned long personality;
int did_exec:1;
unsigned task_dumpable:1;
pid_t pid;
pid_t pgrp;
pid_t tty_old_pgrp;
pid_t session;
pid_t tgid;
int leader;
struct task_struct *real_parent;
struct task_struct *parent;
struct list_head children;
struct list_head sibling;
struct task_struct *group_leader;
struct pid_link pids[PIDTYPE_MAX];
wait_queue_head_t wait_chldexit;
struct completion *vfork_done;
int *set_child_tid;
int *clear_child_tid;
unsigned long rt_priority;
unsigned long it_real_value, it_prof_value, it_virt_value;
unsigned long it_real_incr, it_prof_incr, it_virt_incr;
struct timer_list real_timer;
struct tms times;
struct tms group_times;
unsigned long start_time;
long per_cpu_utime[NR_CPUS], per_cpu_stime[NR_CPUS];
unsigned long min_flt, maj_flt, nswap, cmin_flt, cmaj_flt,
cnswap;
int swappable:1;
uid_t uid,euid,suid,fsuid;
gid_t gid,egid,sgid,fsgid;
int ngroups;
gid_t groups[NGROUPS];
kernel_cap_t   cap_effective, cap_inheritable, cap_permitted;
int keep_capabilities:1;
struct user_struct *user;
struct rlimit rlim[RLIM_NLIMITS];
unsigned short used_math;
char comm[16];
int link_count, total_link_count;
struct tty_struct *tty;
unsigned int locks;
struct sem_undo *semundo;
struct sem_queue *semsleeping;
struct thread_struct thread;
struct fs_struct *fs;
struct files_struct *files;
struct namespace *namespace;
struct signal_struct *signal;
struct sighand_struct *sighand;
sigset_t blocked, real_blocked;
struct sigpending pending;
unsigned long sas_ss_sp;
size_t sas_ss_size;
int (*notifier)(void *priv);
void *notifier_data;
sigset_t *notifier_mask;
void *tux_info;
void (*tux_exit)(void);
      u32 parent_exec_id;
      u32 self_exec_id;
spinlock_t alloc_lock;
        spinlock_t switch_lock;
void *journal_info;
unsigned long ptrace_message;
siginfo_t *last_siginfo;
};
```

Figure by Matt Welsh, Harvard University.

Life cycle of a process

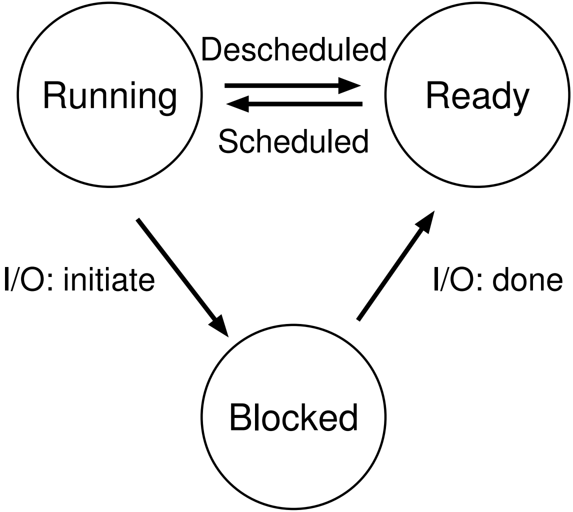
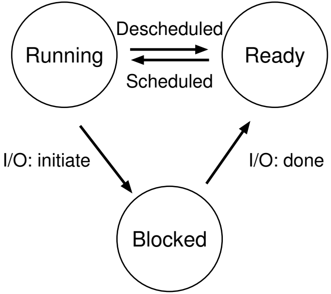


Figure from: OS in three easy pieces

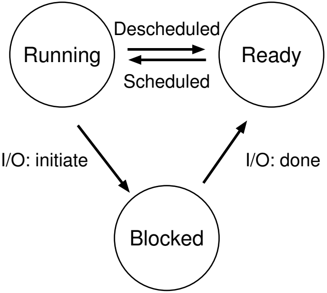
Two processes running, no I/O

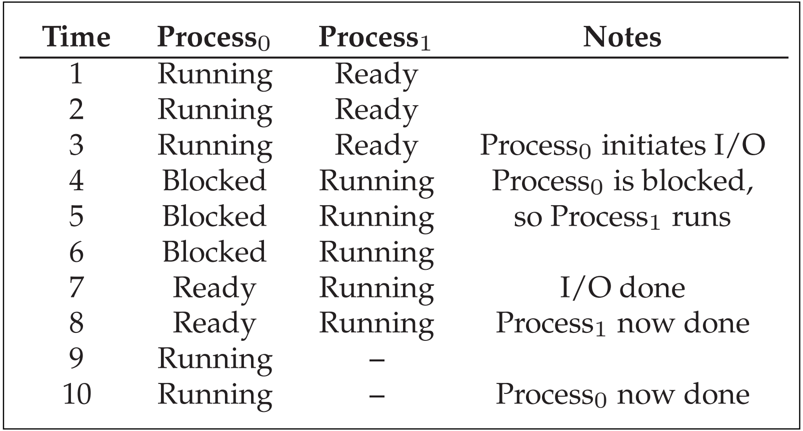


Time	Process$_0$	Process$_1$	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	
4	Running	Ready	Process$_0$ now done
5	–	Running	
6	–	Running	
7	–	Running	
8	–	Running	Process$_1$ now done

Figure from: OS in three easy pieces

Two processes running, with I/O





Time	Process_0	Process_1	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	Process_0 initiates I/O
4	Blocked	Running	Process_0 is blocked,
5	Blocked	Running	so Process_1 runs
6	Blocked	Running	
7	Ready	Running	I/O done
8	Ready	Running	Process_1 now done
9	Running	–	
10	Running	–	Process_0 now done

Figure from: OS in three easy pieces

Process states (Unix)

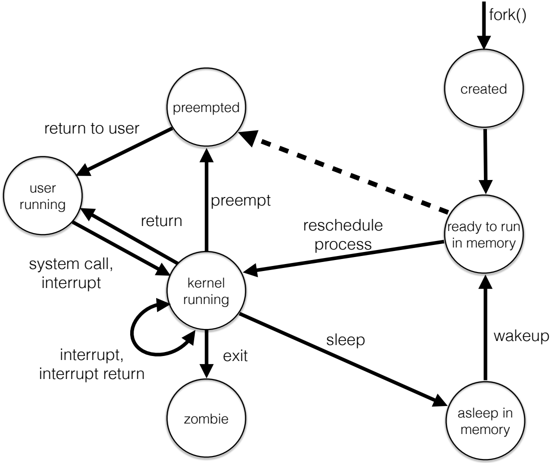


Figure adapted from Stallings' book

Created: Process is newly created but it is not ready to run yet.

Preempted: Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.

Zombie: Process is no longer exists, but it leaves a record for its parent process to collect.

Process states (Unix) without hard drive

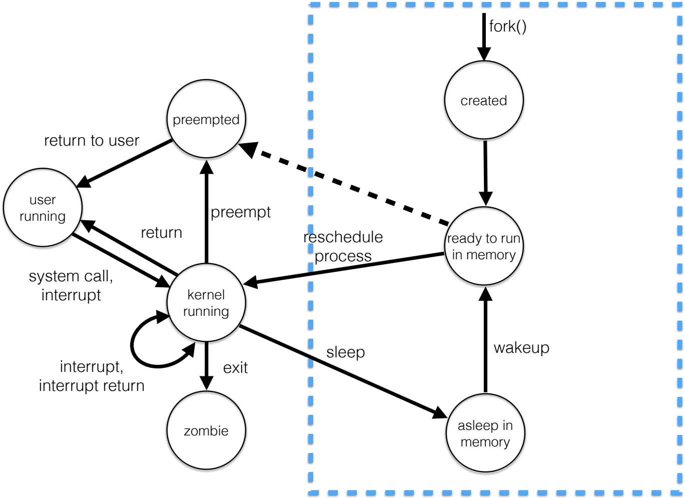


Figure adapted from Stallings' book

Process states (Unix) with hard drive

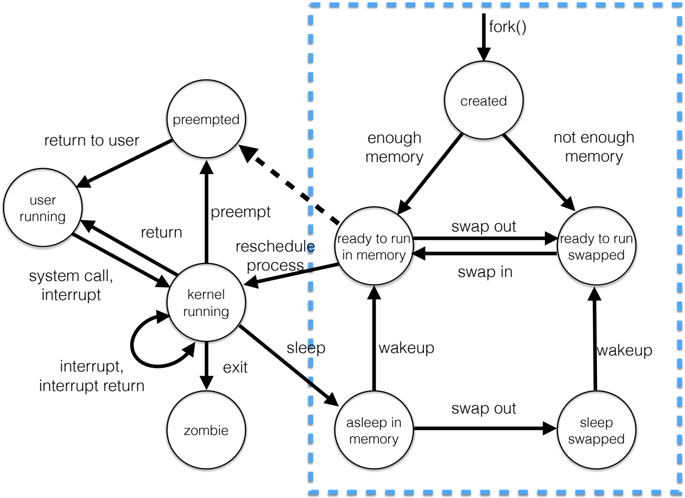
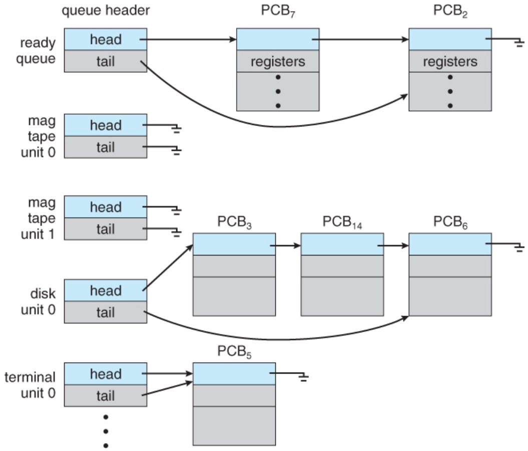
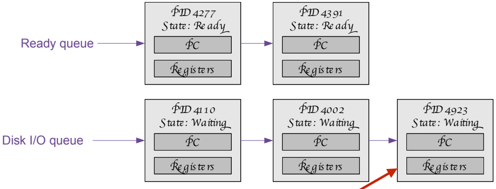


Figure adapted from Stallings' book

Ready queue and various I/O queue: process waiting



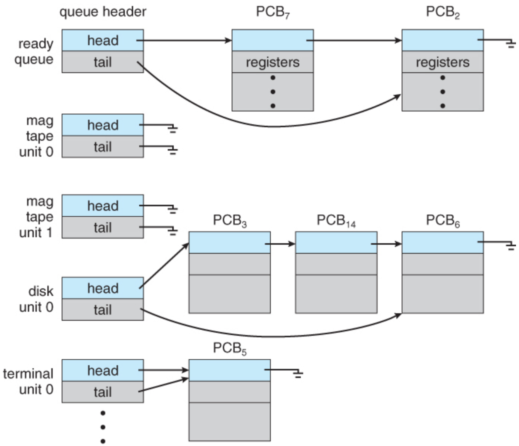
Adapted from Silberschatz, Galvin, and Gagne, 2009.



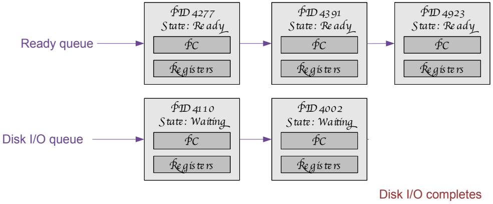
Process waiting

- OS maintains a set of queues
- Each PCB is queued on a state queue based on the process' current state.
- As processes change states, PCBs are unlinked from one queue and linked into another.

Ready queue and various I/O queue: process moved to ready



Adapted from Silberschatz, Galvin, and Gagne, 2009.



- OS maintains a set of queues
- Each PCB is queued on a state queue based on the process’ current state.
- As processes change states, PCBs are unlinked from one queue and linked into another.