

Processes -1-

A program is a sequence of instructions.

A process is

- 'a program in action' (simple view)
- "A sequence of states, resulting from the action of a set of instructions on the states as they develop"

(complex view)

- initial state of computer,
- execute an instruction,
 - a value somewhere changes
- the computer is in a new state.

Process = sum of these states

Processes -2-

not necessarily a 1 to 1 correspondence between a program and a process.

- A process can have a one to many relationship with programs.
- one program can be involved in many processes.
- A process may have many threads.
 - eg a file server

Processes -3- Users && O/S

The operating system may be a number of processes:

- Process management
- Memory management
- File management
- others

One process per user,
everything runs as part of that process

or

a new process for every command and program (unix)

Processes -4- Asynchronous events

Each source of asynchronous events will have its own process.

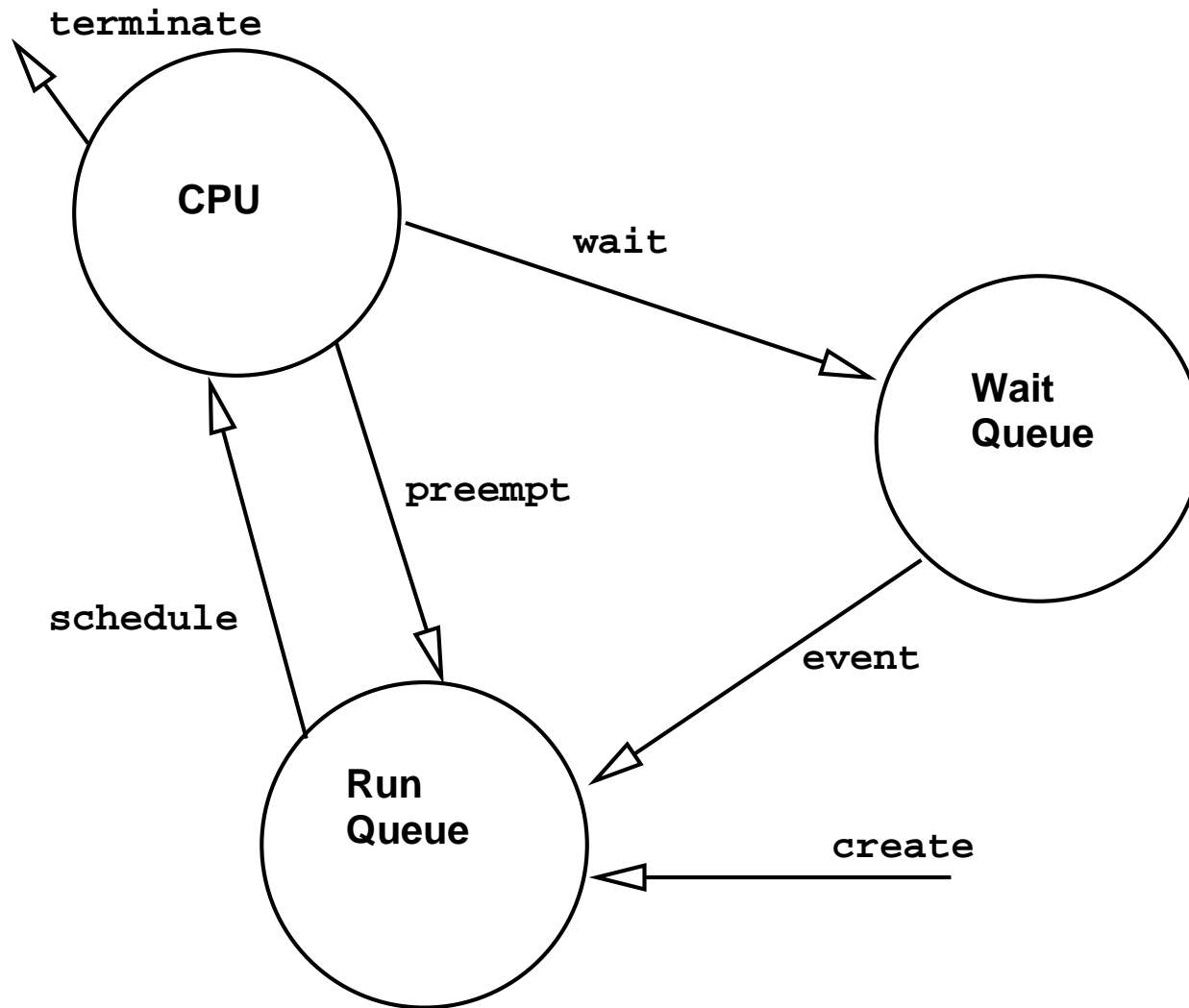
Asynchronous events are

- independent of the currently executing program
- occur at unpredictable times
- may have their own process
 - eg network interface

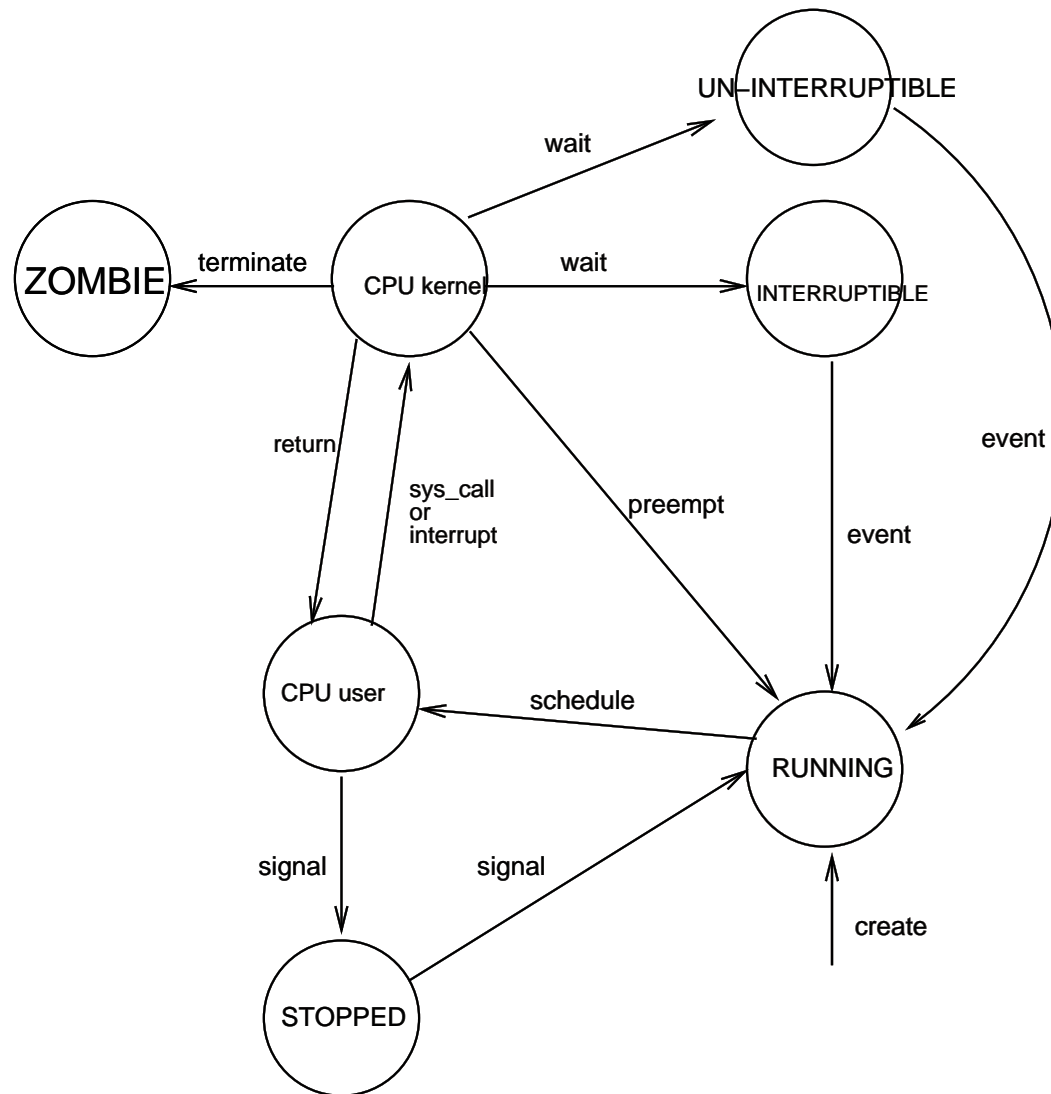
Unix has very few system processes

- most code executes in user context
- either in user mode
 - or, during a system call,
- in kernel mode

Processes : Lifecycle



Processes : Lifecycle



Processes -5a- The Process Table

```
USER  PID %CPU %MEM  VSZ  RSS TTY  STAT  START  TIME  COMMAND
root   1  0.0  0.0  588  72 ?   S   Oct10  0:03  init [3]
root   2  0.0  0.0   0   0 ?   S   Oct10  0:00  [keventd]
root   3  0.0  0.0   0   0 ?   SN  Oct10  0:00  [ksoftirqd_CPU0]
root   4  0.0  0.0   0   0 ?   S   Oct10  0:11  [kswapd]
root   5  0.0  0.0   0   0 ?   S   Oct10  0:00  [bdflush]

ngunton 72  0.0  1.4 19732 7684 pts/0 S   Oct17  0:00  xmms
ngunton 86  0.0  0.4  3692 2172 pts/0 S   Oct17  0:00  xpostit+
ngunton 87  0.0  2.2 15064 11432 tty1 S   Oct17  0:14  /usr/bin/emacs
ngunton 89  0.0  0.1  2408  844 pts/0 R+  08:40  0:00  ps -aux
ngunton 99  0.0  0.1  1792  724 pts/8 T   08:51  0:00  man ps
ngunton 841 0.0  0.0   0   0 pts/1 Z  11:32  0:00  [mwf] <defunct>
```

Status:

S = Sleeping (interruptible), D = Sleeping (uninterruptible)

R = Running or runnable, T = sTopped on signal

X = Dead (shouldn't be seen), Z = Zombie

N = Nice, + = foreground proc group, l = multi-threaded

< = High priority (not nice), s = session leader

Processes -5- Representation

Data structure to represent a process

- known as
 - process descriptor
 - Task Control Block (tcb)
 - Process Control Block (pcb)

Design Problems:

- How do we hold or represent a process in memory?
 - fixed sized tables, or
 - linked lists?

Processes -6-

Fixed size tables

■ Pro:

- index directly into table using process id
- very quick to find process to run.

■ Con:

- fixed limit on number of processes
- wasted space.

Linked lists

■ Pro:

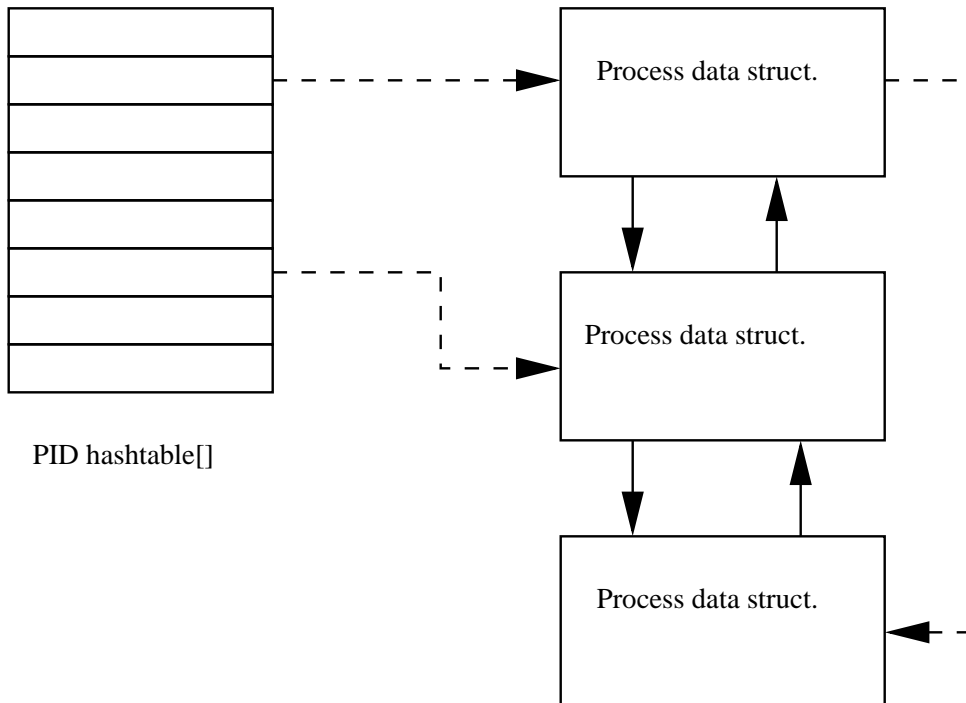
- allocate memory dynamically, recover when process finished

■ Con:

- can't index, must do linear search, time intensive

Processes -7- In GNU/Linux

Linux combines both approaches in the way it manages process tables



Processes -8- PCB contents

What should a process control block contain?

```
struct task_struct {  
  
    volatile long        state;  
    long                counter, priority;  
    struct task_struct   *next_task, *prev_task;  
    struct task_struct   *next_run, *prev_run;  
    int                 exit_code, exit_signal;  
    int                 pid;  
    struct task_struct   *parent, *last_child;  
    struct wait_queue    *wait_childdexit;  
    struct task_struct   *pidhash_next;  
    unsigned long        sched_policy;  
    struct tms           times;  
    unsigned long        start_time;  
    unsigned short       uid, gid;  
    struct thread_struct register_set;  
    struct files_struct  *files;  
    struct mm_struct     *mm;  
    struct signal_struct *sigs;  
    sigset_t            signal, blocked;  
}
```

Processes -9- Process creation

In Unix the fork() system call is used;

```
printf("before the fork\n");  
fork();  
printf("after the fork\n");
```

results in

```
before the fork  
after the fork  
after the fork
```

In Windows the call CreateProcess() does a similar task.
But it takes 9 parameters.