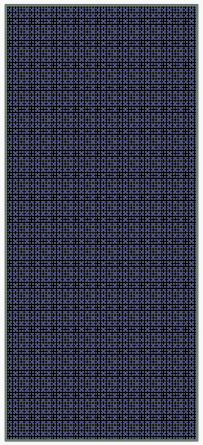
PROCESS

DR. PADMAJA JOSHI



PROCESS

- *Program in execution*

(process id, code, data, register values, pc value)

- *One program may have multiple processes*
- *At any given point in time, only one process is running in a single CPU environment*

PROCESSES

- *Linux gives an ID to every process which is called as PID or processID.*
- *Every process has a unique **pid**.*
- *It is a 5 digit ID number.*

TYPES OF PROCESSES

- *Foreground Processes-*
  - *Gets input from keyboard and outputs on screen;*
  - *Usually every process that we start runs in the foreground;*
  - *When a program is running in foreground, then no other process can run in the foreground as the I/O keyboard and screen are in control of that process;*
  - *Thus the prompt comes back for another process once the current process finishes and hands over the control of IO devices to the OS;*

TYPES OF PROCESSES

- *Background Processes*
  - *It runs without the control of keyboard; If it needs keyboard then it might have to wait;*
  - *When you want a process to run in the background then add & at the end of the command;*

- *When you want to check the current running processes use **ps** command;*
- *It displays PID, PPID, UID, PGID, TTY, STIME, COMMAND;*
- *Has possible parameters*
  - *-a information about all the users*
  - *-x info. About the processes without terminals*
  - *-u additional info like –f*
  - *-e extended info*

PROCESS IN UNIX/LINUX

- *Every process has Process Identification Number (PID), Parent Process Identification Number (PPID)*
- *The first process : sh; ksh;bash;csh*
- *Shells PID is stored in $$*
- *Activities on processes*
  - *Creation*
  - *Management*
  - *Termination*

- *To stop a process use command "**kill <pid>**"*
- *if normal kill command does not kill the process you can use command "**kill -9**"*

PARENT AND CHILD PROCESSES

- *Every process has a parent ID (ppid) along with its own id (pid);*
- *Majority of the processes have shell as their parent;*
- *Parent of all the processes is "init";*
- *Zombie and Orphan Processes*
  - *Usually a child process gets terminated first;*
  - *When a child process gets killed, parent is informed with command "SIGCHLD"*

PARENT AND CHILD PROCESSES

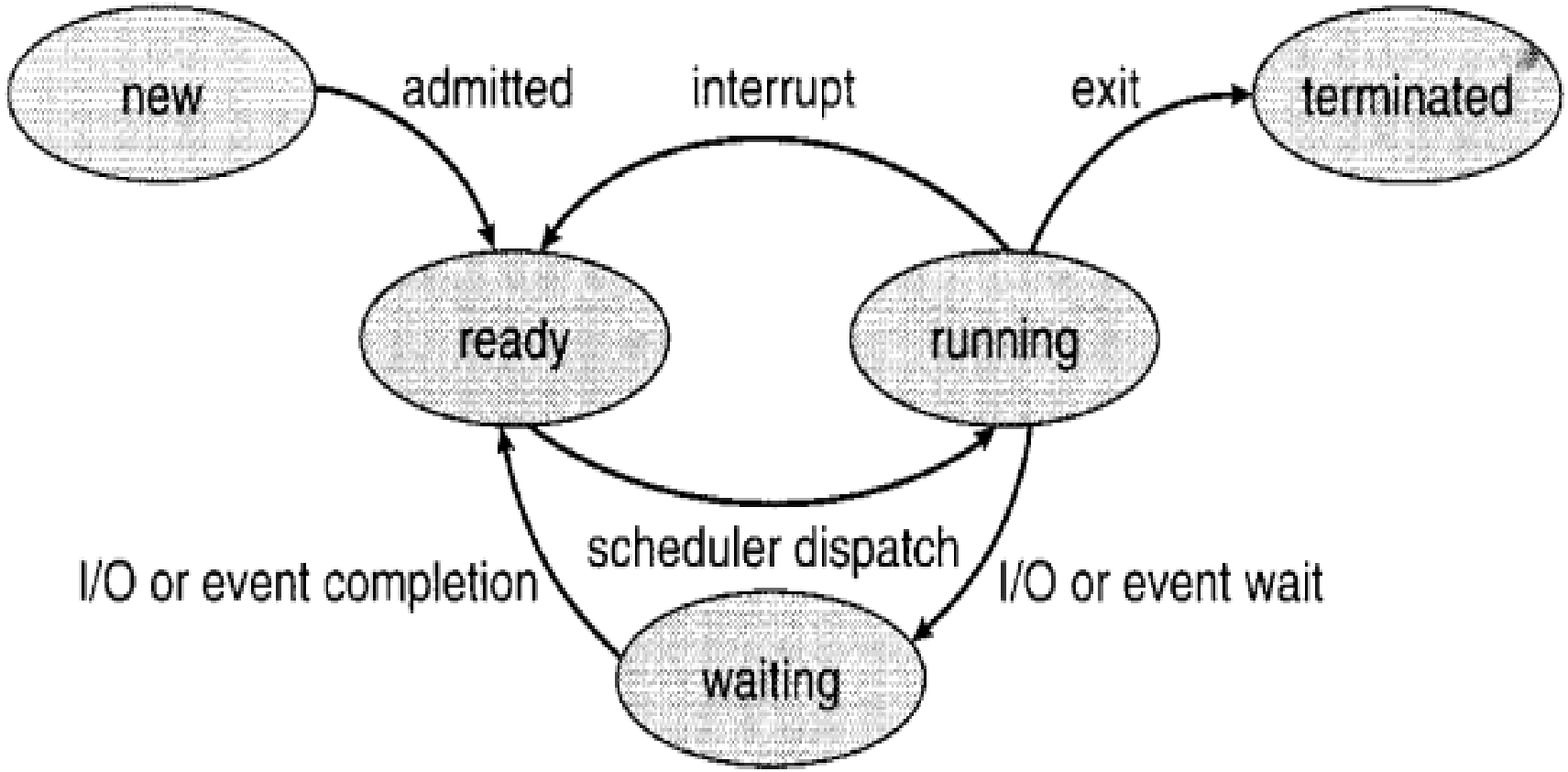
- *Sometimes when a parent process gets killed or terminated before the child process then "init" becomes the parent of the **orphaned processes**.*
- *Processes that have completed their execution but are still present in the system and not dead are called as "**zombie processes**". Such processes are displayed with **z state** in ps command.*

- *Daemon Processes*
  - *System related background processes that run usually with root permission.*

PARENT AND CHILD PROCESSES

- *Daemon has no controlling terminal;*
- *It cannot open **/dev/tty**; on command "ps" you will find "**?**" in **TTY** field of all daemon processes;*

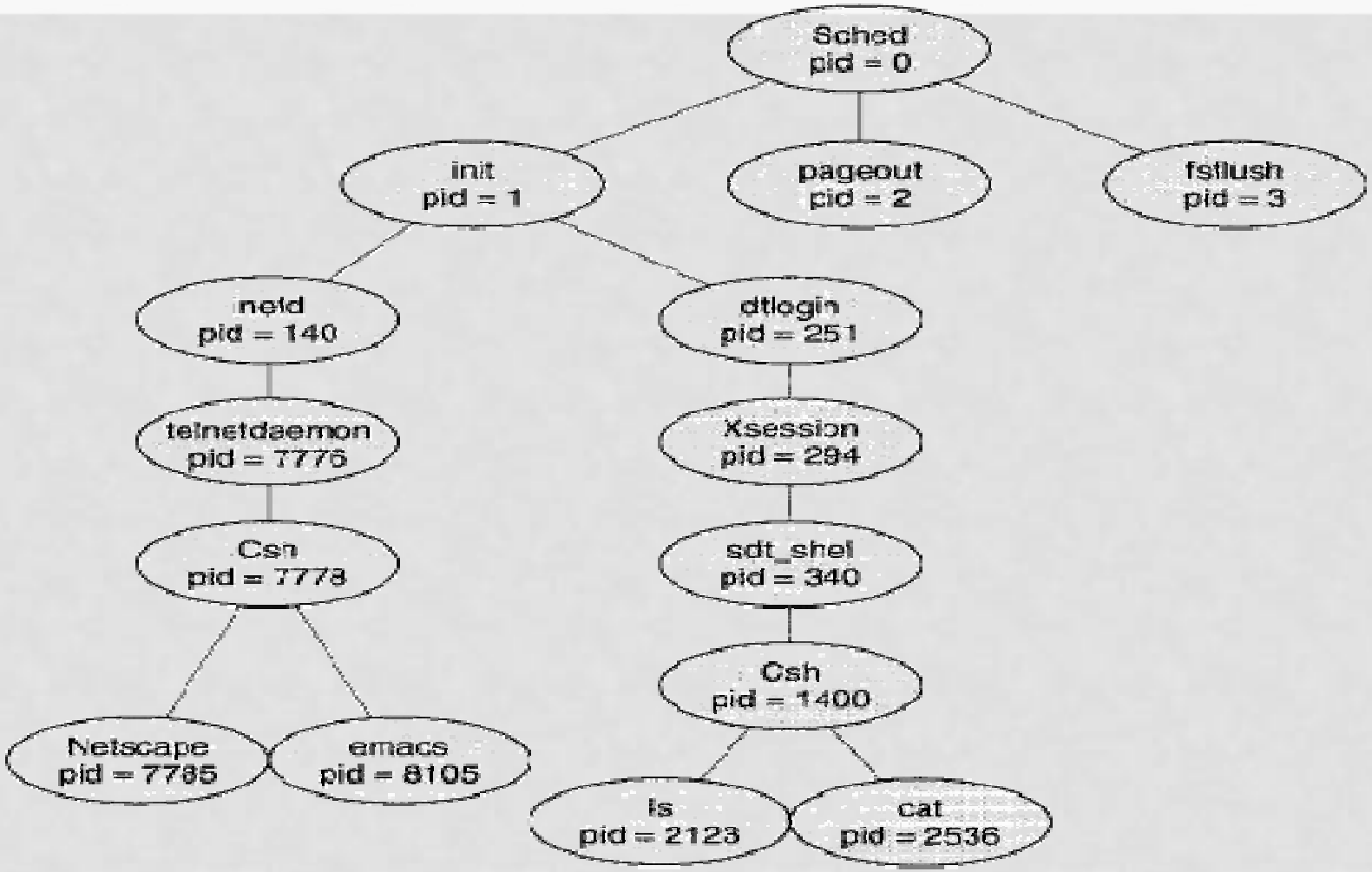
PROCESS STATE



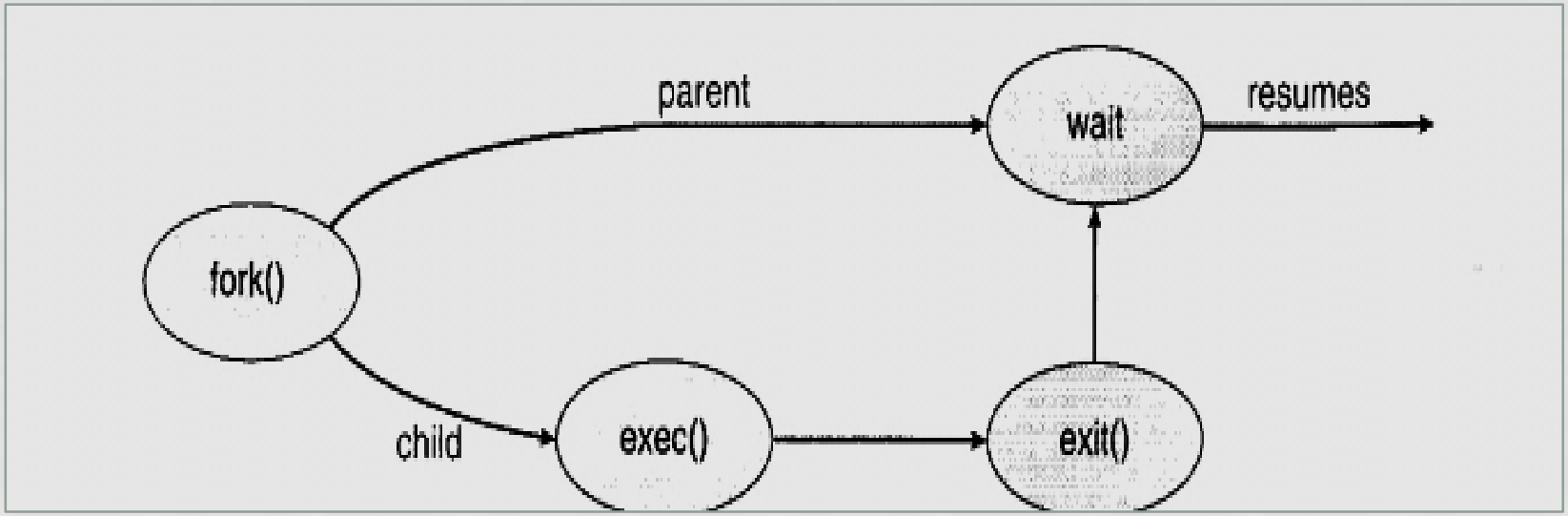
OPERATIONS ON PROCESSES

- Creation
  - Parent process
  - Child process
  - Process tree

PROCESS TREE OF SOLARIS



PROCESS CREATION



CREATING A NEW PROCESS IN LINUX

- System call used to create a new process in Linux is fork()
  - Returns 0 to the child process if the operation is successful and returns pid of the child process to the parent process;
  - It returns negative value when cannot create a process;

GETTING PID AND PPID

- ps : gives pid
- ps –f : gives ppid

FORK() FOR CREATING PROCESS

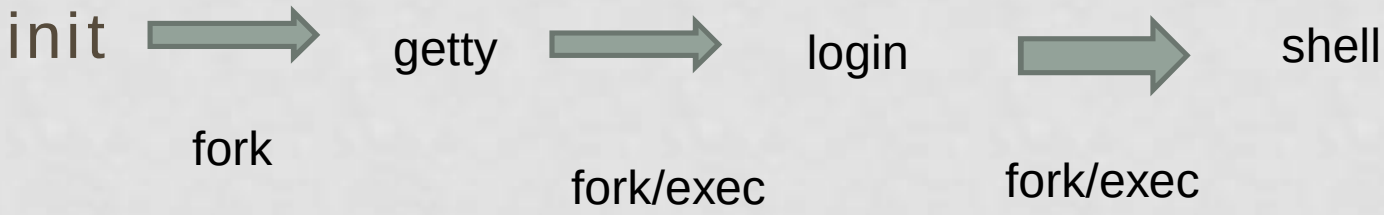
pid_t fork(void)

- Returns PID of a child to its parent
- 0 to child

Note that the changes made in a child process are not visible in parent process

Demonstrate with fork.c

HOW SHELL IS CREATED?



Three commands /system calls related to Process creation are
- fork,
- exec,
- wait

STATES IN LINUX

/* in tsk->exit_state */

#define EXIT_ZOMBIE 16 - the process is exiting but has not yet been waited for by its parent

#define EXIT_DEAD 32 - the process has exited and has been waited for

/* in tsk->state again */

#define TASK_NONINTERACTIVE 64

REPRESENTATION OF PROCESS IN LINUX

- Called as process descriptor
- Stored in a structure called **struct task_structs** declared in **linux/sched.h**

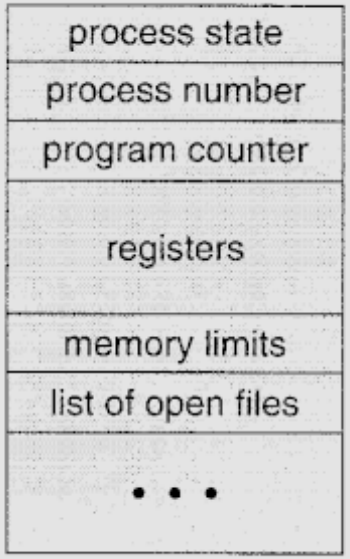
PARTIAL CODE OF STRUCT TASK_STRUCT

```
701 struct task_struct {
702 volatile long state; /* -1 unrunnable, 0 runnable, >0
stopped */
703 struct thread_info *thread_info; . .

767 /* PID/PID hash table linkage. */
 768 struct pid pids[PIDTYPE_MAX]; . .
798 char comm[TASK_COMM_LEN]; /* executable
name excluding path
```

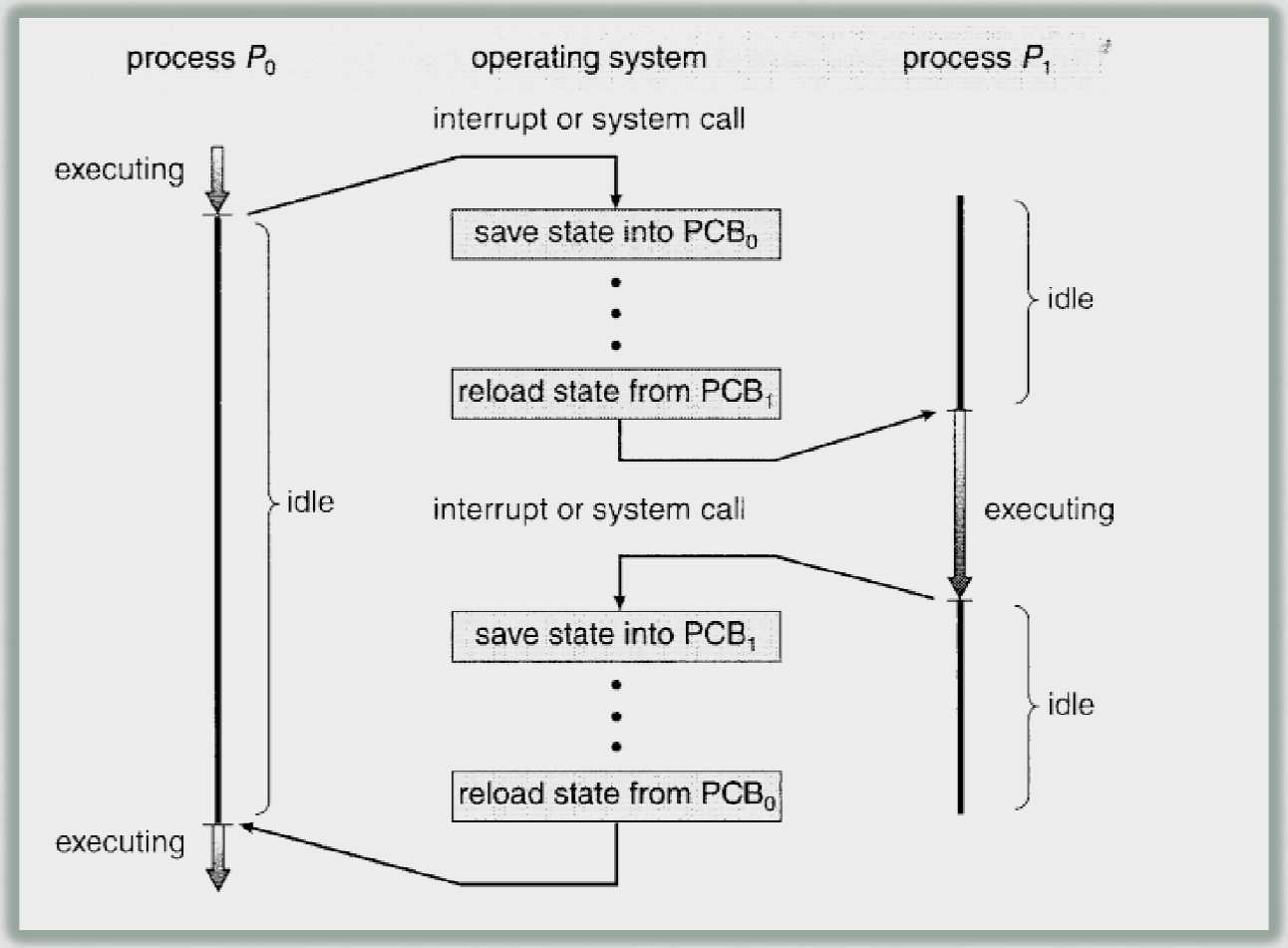
PROCESS CONTROL BLOCK (PCB)

Also called as Task Control Block



Maintains information such as priority, Program status register(PSR), event information, memory and resource information, PCB pointer

RUNNING MULTIPLE PROCESSES



CONTEXT SWITCH

- *Interrupt cause OS to change CPU from its current task to run a kernel routine*
- *What is **context switch**?*

In order to provide multi-tasking in single CPU environment context switching is used. It stores the context or state of the running process so that it can be reloaded/resumed whenever required.

REFERENCES

[1]. D.M. Dhamdhere," Systems Programming and Operating Systems", 2e, TMH, 2001

[2]. http://www.cs.columbia.edu/~junfeng/10sp-w4118/lectures/l07-proc-linux.pdf

[3]. http://linuxgazette.net/133/saha.html

[4].William Stallings "Operating Systems Internals and Design Principles", 6e, Peason Education, 2009