

LECTURE - III PROCESSES

Tevfik Koşar

University at Buffalo
September 6th, 2011

Roadmap

- Processes
 - Basic Concepts
 - Process Creation
 - Process Termination
 - Context Switching
 - Process Queues
 - Process Scheduling
 - Interprocess Communication



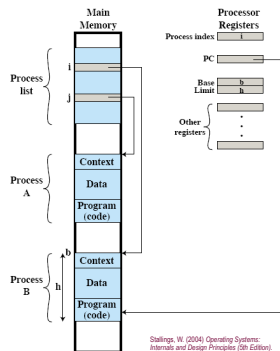
2

Process Concept

- a **Process** is a **program in execution**;

➤ A process image consists of three components

1. an executable **program**
2. the associated **data** needed by the program
3. the execution **context** of the process, which contains all information the O/S needs to manage the process (ID, state, CPU registers, stack, etc.)



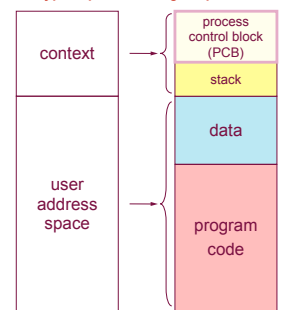
Typical process image implementation

Stallings, W. (2004) Operating Systems: Internals and Design Principles (3rd Edition).

Process Control Block

➤ The Process Control Block (PCB) Typical process image implementation

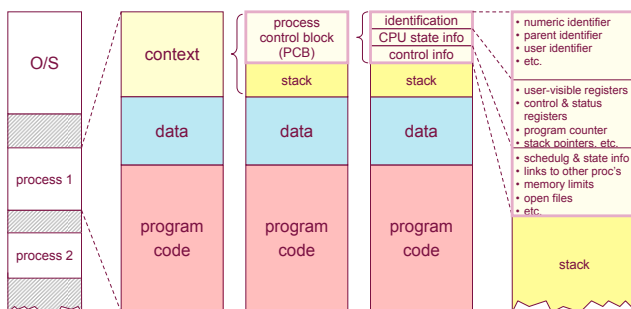
- ✓ is included in the context, along with the stack
- ✓ is a "snapshot" that contains all necessary and sufficient data to restart a process where it left off (ID, state, CPU registers, etc.)
- ✓ is one entry in the operating system's **process table** (array or linked list)



4

Process Control Block

➤ Example of process and PCB location in memory

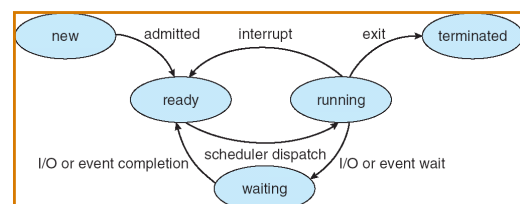


Illustrative contents of a process image in (virtual) memory

5

Process State

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



6

Process Creation

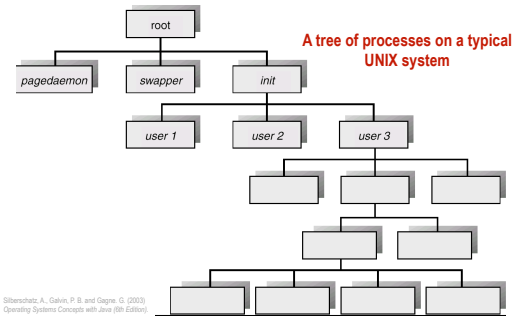
Some events that lead to process creation (enter)

- all cases of process spawning
- ✓ the system boots
 - when a system is initialized, several background processes or "daemons" are started (email, logon, etc.)
 - ✓ a user requests to run an application
 - by typing a command in the CLI shell or double-clicking in the GUI shell, the user can launch a new process
 - ✓ an existing process spawns a child process
 - for example, a server process (print, file) may create a new process for each request it handles
 - the *init* daemon waits for user login and spawns a shell
 - ✓ a batch system takes on the next job in line

7

Process Creation

Process creation by spawning



8

Process Creation

Implementing a shell command interpreter by process spawning

```

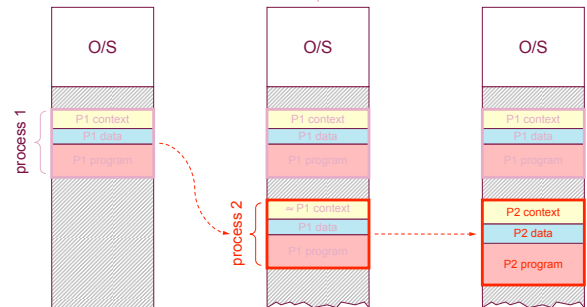
...
int main(...)
{
    ...
    if ((pid = fork()) == 0)           // create a process
    {
        fprintf(stdout, "Child pid: %i\n", getpid());
        err = execvp(command, arguments); // execute child
                                           // process
        fprintf(stderr, "Child error: %i\n", errno);
        exit(err);
    }
    else if (pid > 0)                 // we are in the
    {                                  // parent process
        fprintf(stdout, "Parent pid: %i\n", getpid());
        pid2 = waitpid(pid, &status, 0); // wait for child
        ...
    }
    ...
    return 0;
}

```

9

Process Creation

1. Clone child process
✓ `pid = fork()`
2. Replace child's image
✓ `execve(name, ...)`



10

Fork Example 1

```

#include <stdio.h>

main()
{
    int ret_from_fork, mypid;

    mypid = getpid();           /* who am i? */
    printf("Before: my pid is %d\n", mypid); /* tell pid */

    ret_from_fork = fork();

    sleep(1);
    printf("After: my fork returns pid : %d, said %d\n",
           ret_from_fork, getpid());
}

```

11

Fork Example 2

```

#include <stdio.h>

main()
{
    fork();
    fork();
    fork();
    printf("my pid is %d\n", getpid() );
}

```

How many lines of output will this produce?

12

Process Termination

➤ Some events that lead to process termination (exit)

- ✓ regular completion, with or without error code
 - process-triggered
 - the process voluntarily executes an `exit(err)` system call to indicate to the O/S that it has finished
- ✓ fatal error (uncatchable or uncaught)
 - O/S-triggered (following system call or preemption)
 - service errors: no memory left for allocation, I/O error, etc.
 - total time limit exceeded
 - hardware interrupt-triggered
 - arithmetic error, out-of-bounds memory access, etc.
- ✓ killed by another process via the kernel
 - software interrupt-triggered
 - the process receives a `SIGKILL` signal
 - in some systems the parent takes down its children with it

13

Process Pause/Dispatch

➤ Some events that lead to process pause / dispatch

- ✓ I/O wait
 - O/S-triggered (following system call)
 - a process invokes an I/O system call that blocks waiting for the I/O device: the O/S puts the process in "Waiting" mode and dispatches another process to the CPU
- ✓ preemptive timeout
 - hardware interrupt-triggered (timer)
 - the process receives a timer interrupt and relinquishes control back to the O/S dispatcher: the O/S puts the process in "Ready" mode and dispatches another process to the CPU
 - not to be confused with "total time limit exceeded", which leads to process termination

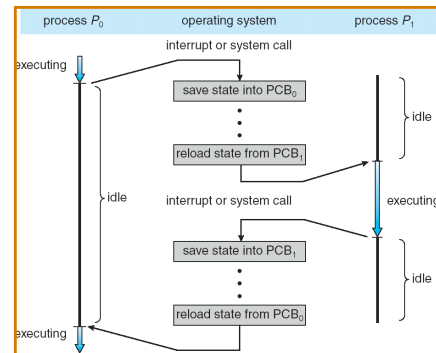
14

Process "Context" Switching

- 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
- Switching time is dependent on hardware support

15

CPU Switch From Process to Process



16

Process "Context" Switching

➤ How does a full process switch happen, step by step?

1. save CPU context, including PC and registers (*the only step needed in a simple mode switch*)
2. update process state (to "Ready", "Blocked", etc.) and other related fields of the PCB
3. move the PCB to the appropriate queue
4. select another process for execution: this decision is made by the CPU scheduling algorithm of the O/S
5. update the PCB of the selected process (state = "Running")
6. update memory management structures
7. restore CPU context to the values contained in the new PCB

17

Process "Context" Switching

➤ What events trigger the O/S to switch processes?

- ✓ **interrupts** — external, asynchronous events, independent of the currently executed process instructions
 - clock interrupt → O/S checks time and may block process
 - I/O interrupt → data has come, O/S may unblock process
 - memory fault → O/S may block process that must wait for a missing page in memory to be swapped in
- traps { ✓ **exceptions** — internal, synchronous (but involuntary) events caused by instructions → O/S may terminate or recover process
- traps { ✓ **system calls** — voluntary synchronous events calling a specific O/S service → after service completed, O/S may either resume or block the calling process, depending on I/O, priorities, etc.

18

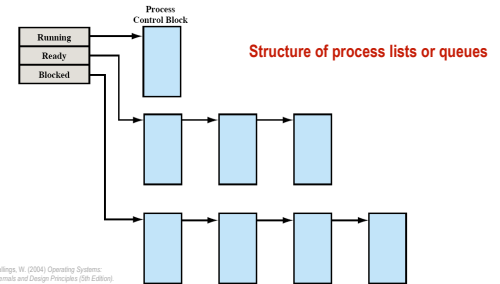
Process Scheduling Queues

- **Job queue** - set of all jobs 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

19

Process Queues

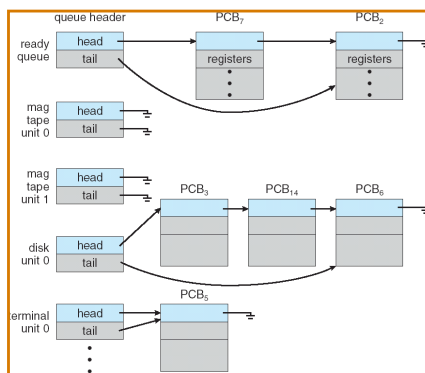
- The process table can be split into per-state queues
 - ✓ PCBs can be linked together if they contain a pointer field



Stallings, W. (2004) Operating Systems: Internals and Design Principles (2th Edition)

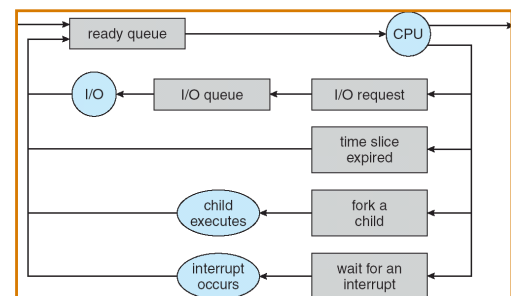
20

Ready Queue And Various I/O Device Queues



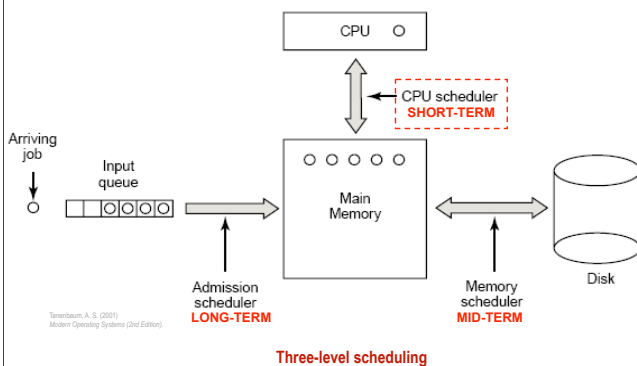
21

Representation of Process Scheduling



22

Three Level CPU Scheduling



Tanenbaum, A. S. (2001) Modern Operating Systems (2nd Edition)

Three-level scheduling

23

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

24

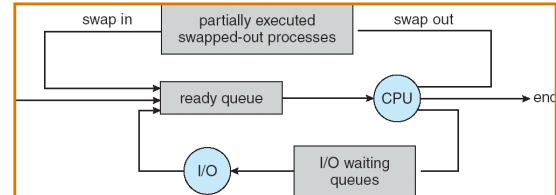
Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (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 schedulers need to make careful decision

25

Addition of Medium Term Scheduling

- In time-sharing systems: remove processes from memory “temporarily” to reduce degree of multiprogramming.
- Later, these processes are resumed → **Swapping**



26

Cooperating Processes

- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
 - Information sharing
 - Computation speed-up
 - Modularity
 - Convenience
- Disadvantage
 - Synchronization issues and race conditions

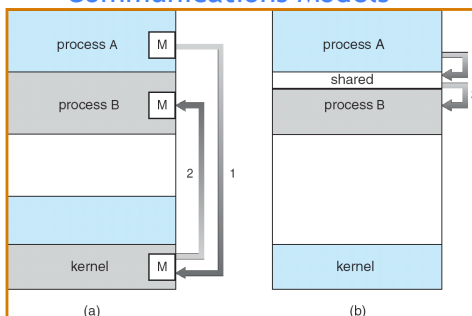
27

Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions
- **Shared Memory:** by using the same address space and shared variables
- **Message Passing:** processes communicate with each other without resorting to shared variables

28

Communications Models



a) Message Passing

b) Shared Memory

29

Message Passing

- Message Passing facility provides two operations:
 - `send(message)` - message size fixed or variable
 - `receive(message)`
- If *P* and *Q* wish to communicate, they need to:
 - establish a *communication link* between them
 - exchange messages via `send/receive`
- Two types of Message Passing
 - direct communication
 - indirect communication

30

Message Passing - direct communication

- Processes must name each other explicitly:
 - `send(P, message)` - send a message to process P
 - `receive(Q, message)` - receive a message from process Q
- Properties of communication link
 - Links are established automatically
 - A link is associated with exactly one pair of communicating processes
 - Between each pair there exists exactly one link
 - The link may be unidirectional, but is usually bi-directional
- Symmetrical vs Asymmetrical direct communication
 - `send(P, message)` - send a message to process P
 - `receive(id, message)` - receive a message from any process
- Disadvantage of both: limited modularity, hardcoded

31

Message Passing - indirect communication

- Messages are directed and received from mailboxes (also referred to as ports)
 - Each mailbox has a unique id
 - Processes can communicate only if they share a mailbox
- Primitives are defined as:
 - `send(A, message)` - send a message to mailbox A
 - `receive(A, message)` - receive a message from mailbox A

32

Indirect Communication (cont.)

- Operations
 - create a new mailbox
 - send and receive messages through mailbox
 - destroy a mailbox
- Properties of communication link
 - Link established only if processes share a common mailbox
 - A link may be associated with many processes
 - Each pair of processes may share several communication links
 - Link may be unidirectional or bi-directional

33

Indirect Communication (cont.)

- Mailbox sharing
 - P_1 , P_2 , and P_3 share mailbox A
 - P_1 sends; P_2 and P_3 receive
 - Who gets the message?
- Solutions
 - Allow a link to be associated with at most two processes
 - Allow only one process at a time to execute a receive operation
 - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

34

Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
 - **Blocking send** has the sender block until the message is received
 - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
 - **Non-blocking send** has the sender send the message and continue
 - **Non-blocking receive** has the receiver receive a valid message or null

35

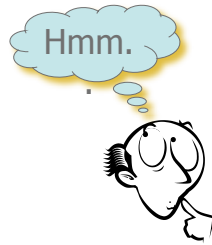
Buffering

- Queue of messages attached to the link; implemented in one of three ways
 1. Zero capacity - 0 messages
Sender must wait for receiver (rendezvous)
 2. Bounded capacity - finite length of n messages
Sender must wait if link full
 3. Unbounded capacity - infinite length
Sender never waits

36

Summary

- Processes
 - Basic Concepts
 - Process Creation
 - Process Termination
 - Context Switching
 - Process Queues
 - Process Scheduling
 - Interprocess Communication



- Next Lecture: Threads
- Reading Assignment: Chapter 3 from Silberschatz.
- HW 1 will be out next class, due 1 week

37

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR

38