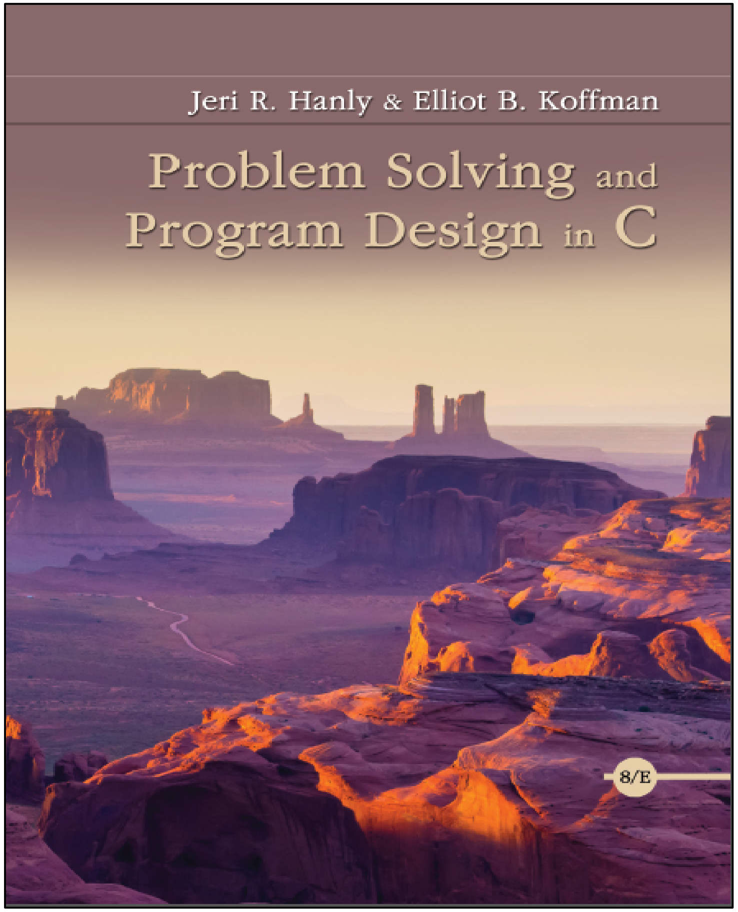
Problem Solving and Program Design in C

Eighth Edition



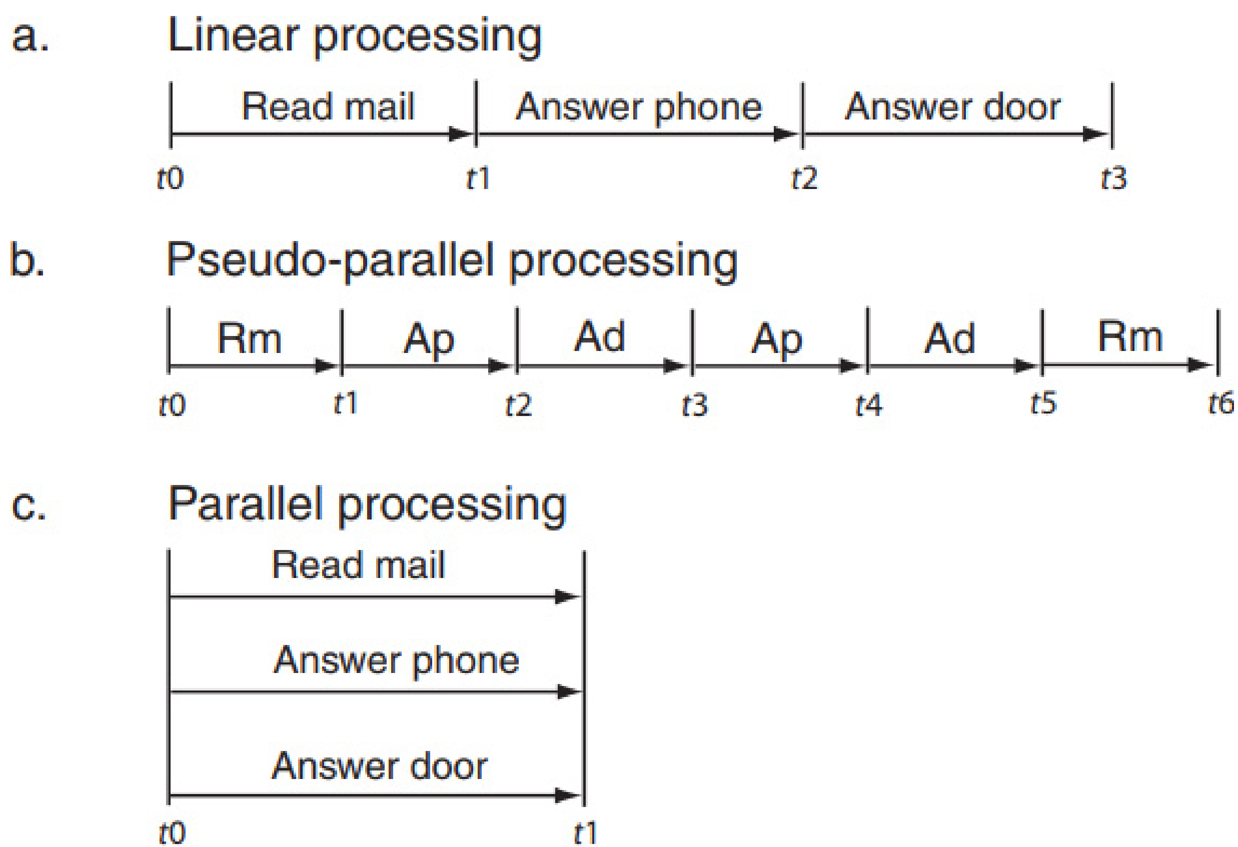
Chapter 14

Multiprocessing Using Processes and Threads

Terminology (1 of 4)

- multitasking
  - dividing a program into tasks that operate independently of one another
- linear programming
  - writing a sequence of program instructions in which each instruction depends on the completion of the previous instruction
- parallel programming
  - execution of multiple programs at the same time

Figure 14.1 Three Modes of Processing



Terminology (2 of 4)

- time-sharing
  – performing parallel programming by allocating to each system user a portion of the available CPU time
- preemptive multitasking
  – stopping the execution of a running program by the hardware interrupt system, allowing another program to access the CPU

Terminology (3 of 4)

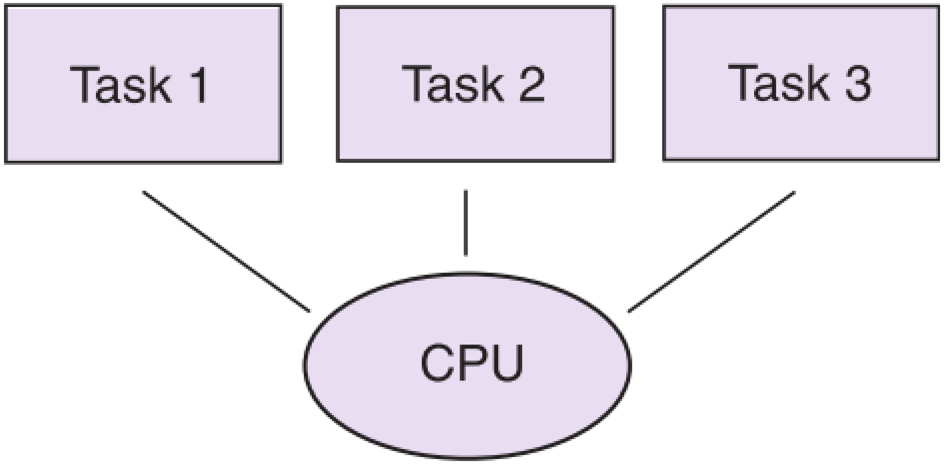
- pseudo-parallelism
  - a situation in which programs appear to be running in parallel at the same time although they are actually taking turns sharing the CPU
- time slice
  - the amount of CPU time allocated to each program in a parallel programming environment

Terminology (4 of 4)

- context switch
  - the process of switching from one process to another accomplished by saving the state information for the currently executing process, which will become idle, and loading the saved state information for a currently idle process, which will resume execution
- concurrent programming
  - writing sets of program instructions that can execute at the same time independently of one another

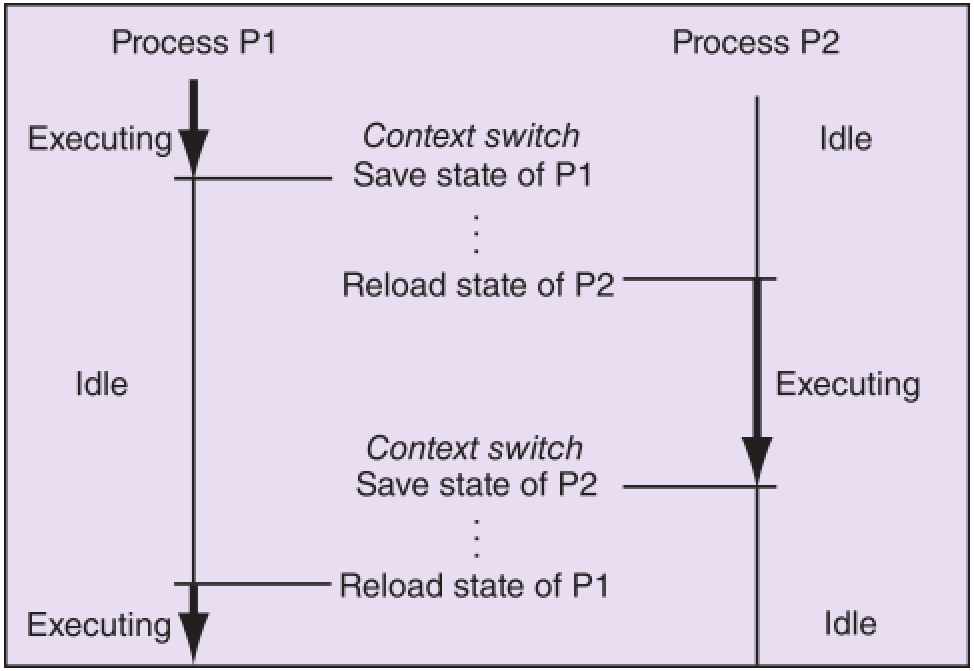
Pearson

Figure 14.2 Preemptive Multitasking



The CPU executes Task 1 for awhile…
Then Task 2…
Then Task 3…
Then Task 1…
Then Task 2…
Then Task 3…
And so on…

Figure 14.3 Context Switching from P1 to P2 to P1



Processes (1 of 2)

- process ID
  - a unique identifier given to a process by the operating system
- child process
  - a new process that is created by a currently executing process (the parent process)
- parent process
  - the currently executing process that has created one or more new child processes

fork()

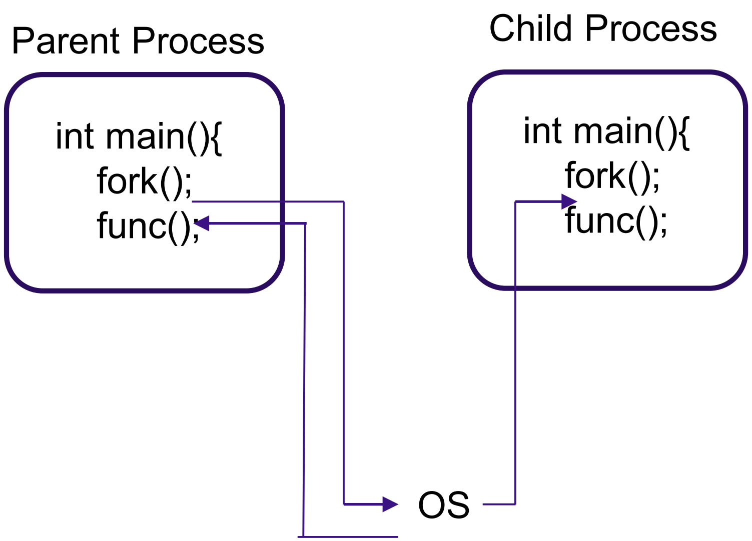
- Fork system call is used for creating a new process, which is called ***child process***, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

- It takes no parameters and returns an integer value. Below are different values returned by fork().

- ***Negative Value***: creation of a child process was unsuccessful.
  Zero: Returned to the newly created child process.
  Positive value: Returned to parent or caller. The value contains process ID of newly created child process. from https://www.geeksforgeeks.org/fork-system-call/

Pearson

fork()



- fork1.c
- fork2.c

https://www.geeksforgeeks.org/fork-system-call/

fork()

A process executes the following code

```
for (i = 0; i < n; i++) fork();
```

The total number of child processes created is

(A) n

(B) $2^n - 1$

(C) 2^n

(D) $2^{(n+1)} - 1$

https://www.geeksforgeeks.org/fork-system-call/

fork()

A process executes the following code

```
for (i = 0; i < n; i++) fork();
```

The total number of child processes created is

(A) n

(B) $2^n - 1$

(C) 2^n

(D) $2^{(n+1)} - 1$

https://www.geeksforgeeks.org/fork-system-call/

ANSWER IS B

fork()

2. Consider the following code fragment:

```
if (fork() == 0) {
    a = a + 5;
    printf("%d, %d\n", a, &a);
}
else {
    a = a -5;
    printf("%d, %d\n", a, &a);
}
```

Let u, v be the values printed by the parent process, and x, y be the values printed by the child process. Which one of the following is TRUE? (GATE-CS-2005)

(A) u = x + 10 and v = y

(B) u = x + 10 and v != y

(C) u + 10 = x and v = y

(D) u + 10 = x and v != y

https://www.geeksforgeeks.org/fork-system-call/

fork()

2. Consider the following code fragment:

```
if (fork() == 0) {
    a = a + 5;
    printf("%d, %d\n", a, &a);
}
else {
    a = a -5;
    printf("%d, %d\n", a, &a);
}
```

Let u, v be the values printed by the parent process, and x, y be the values printed by the child process. Which one of the following is TRUE? (GATE-CS-2005)

(A) u = x + 10 and v = y

(B) u = x + 10 and v != y

(C) u + 10 = x and v = y

(D) u + 10 = x and v != y

fork3.c

https://www.geeksforgeeks.org/fork-system-call/

ANSWER IS C

Table 14.1 Some Process Functions from `unistd.h` and `wait.h`

Function	Purpose: Example	Parameters	Result Type
`fork`	If successful, creates a new process and returns the process ID of the new process (to the parent) and 0 (to the new process). If not successful, returns –1 to the parent. `pid = fork();`	None	`pid_t`
`getpid`	Returns the process id of the calling process. `pid = getpid();`	None	`pid_t`
`wait`	Returns the process id of the next child process to exit. The exit status is written into the memory location pointed to by `status_ptr`. `pid = wait( &status_ptr );`	`int* status_ptr`	`pid_t`

`execl`	Replaces the instructions in the process that is executing with the instructions in the executable file specified by the `path` and `file` arguments. The argument `path` specifies the full path name including the executable file name; the argument `file` specifies just the executable file name; the ellipses indicates that there may be more arguments, but the last argument is always `NULL`.
	
	`execl ("prog.exe", "prog.exe", NULL);`
		`const char *path`
		`const char *file`
		`. . .`
		`NULL`
			`int`

execl1.c, execl2.c, execl3.c, execl4.c

Processes (2 of 2)

- zombie
  - a child process that has exited but whose parent process has not yet retrieved its exit status
- defunct process
  - a child process that has exited but whose parent process has not yet retrieved its exit status

Interprocess Communications and Pipes (1 of 2)

- interprocess communications
  - the exchange of information between processes that are running on the same CPU and that have a common ancestor
- pipe
  - a form of interprocess communications that consists of two file descriptors, one opened for reading and the other opened for writing

Interprocess Communications and Pipes (2 of 2)

- half-duplex pipe
  - a pipe which can send information only in one direction
- full-duplex pipe
  - a pipe which can send information in both directions at the same time

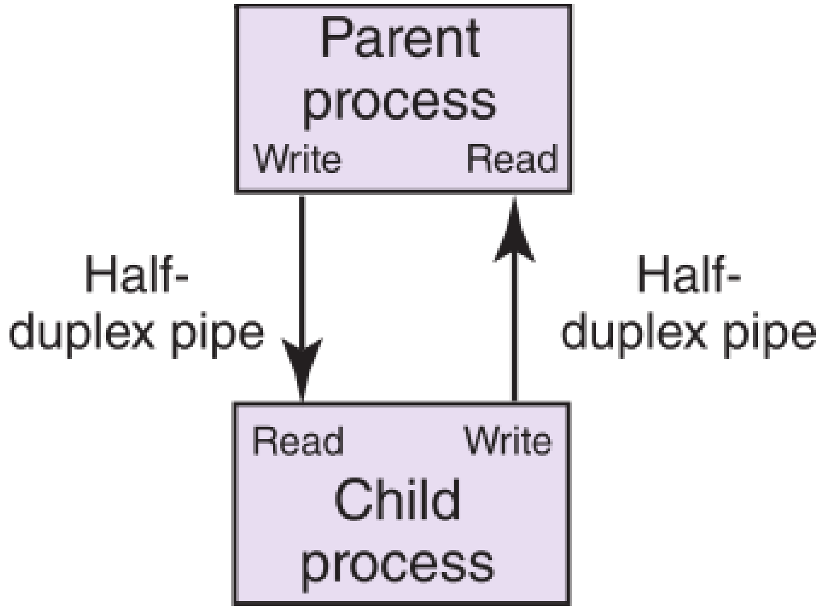
Table 14.2 Some Interprocess Communications Functions from `unistd.h`

Function	Purpose: Example	Parameters	Result Type
`pipe`	If successful, creates a new pipe and returns a value of 0.The read and write file descriptors are written into the array argument. If not successful, returns –1. `pipe (filedes);`	`int filedes[2]`	`int`
`dup2`	If successful, duplicates the file descriptor `oldfiledes` into the file descriptor `newfiledes` and returns `newfiledes`. If not successful, returns –1. `dup2 (oldfiledes, newfiledes);`	`int oldfiledes` `int newfiledes`	`int`
`sleep`	If successful, pauses the program execution for the specified number of seconds and returns 0. If unsuccessful, returns the number of seconds remaining to sleep. `sleep (seconds);`	`unsigned int seconds`	`unsigned int`
`close`	Closes the designated file. Returns 0 if successful; –1 if unsuccessful. `close (oldfiledes)`	`int oldfiledes`	`int`

`read`	Reads `numbytes` bytes from file `oldfiledes` into array `buffer`. Returns the number of bytes read if successful; returns –1 if unsuccessful.
	`read (oldfiledes, buffer, numbytes)`
		`int oldfiledes`
		`void* buffer`
		`size_t numbytes`
			`size_t`

`write`	Writes `numbytes` bytes to file `newfiledes` from array `buffer`. Returns the number of bytes read if successful; returns –1 if unsuccessful.
	`write (newfiledes, buffer, numbytes)`
		`int newfiledes`
		`void* buffer`
		`size_t numbytes`
			`size_t`

Figure 14.4 Interprocess Communications Using Half-duplex Pipes



pipe2.c

Wrap up

- Multitasking is a way for a single user to run many programs at the same time on a single CPU while still allowing the user to maintain control over the CPU.
- Preemptive multitasking is a way to preempt a running program with the hardware interrupt system and instructing the CPU to run another program.
- Concurrent programming involves writing sets of program instructions that can execute at the same time independently of one another.

Copyright