Interprocess Communication

P.C.P. Bhatt

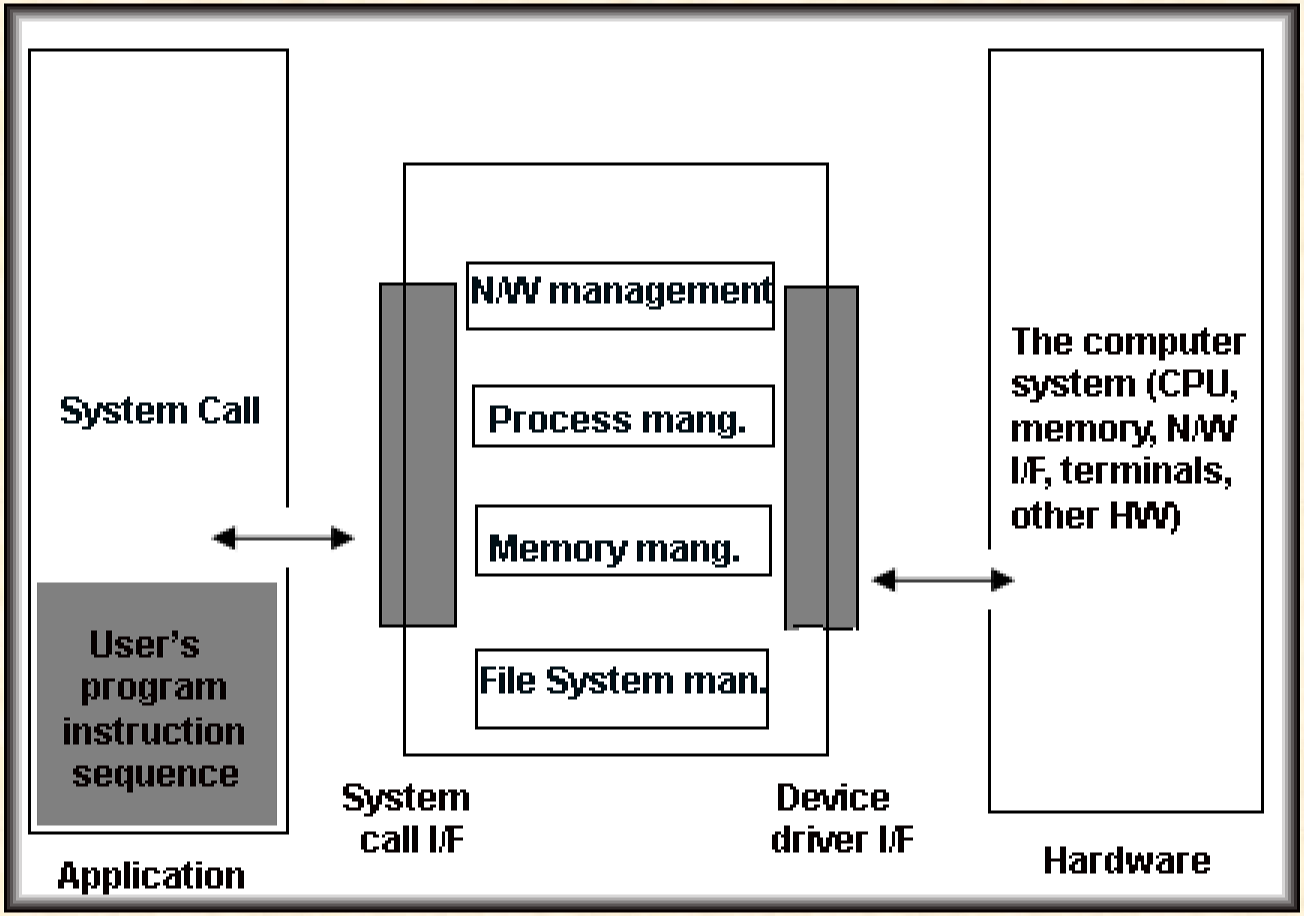
Introduction

- IPC coordinates between computation spread over several processes.
- IPC enables communication amongst process.
- Synchronization amongst processes.
- Need for IPC arises in parallel and distributed processing contexts.

Creating a New Process

- The fork() system call brings in a new process into an existing execution environment.
- *fork()* invokes kernel services for process creation.
- The system call *fork()* spawns a new process which is a copy of the parent process from where it is invoked.

Processing System calls



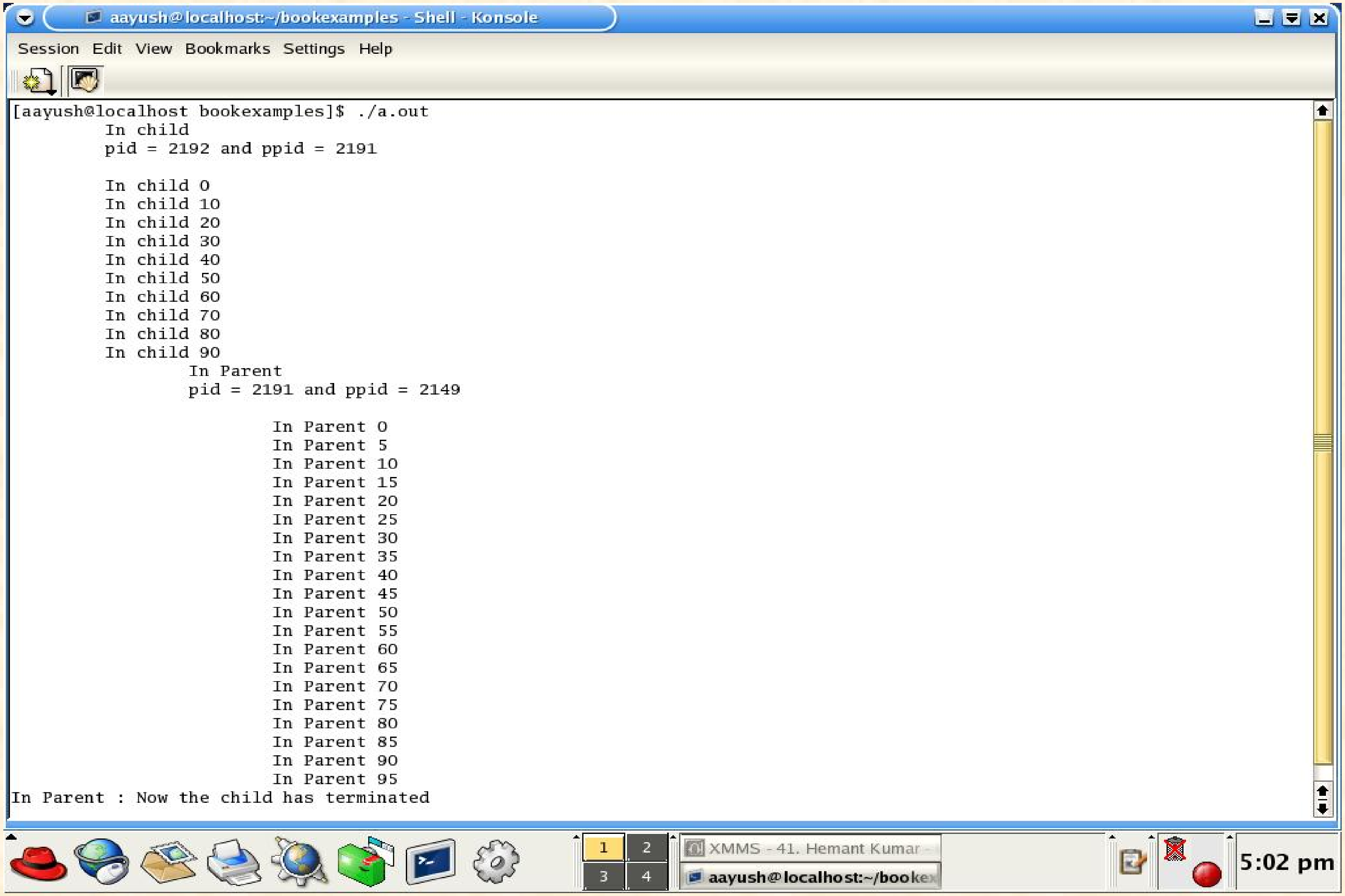
The fork() System Call

- Signals are powerful interprocess communication mechanism.
- Wait and Exit are utilized to have interprocess communication in particular for synchronize activities of the process.
- The return value of system call is utilized to identify when the parent or child is in execution.

Demonstrating use of fork system call

```
//The Program: Demonstration of the use of fork() system call
int main()
{
    int i,j;
    if (fork()) /*must be parent */
    {
            printf("\t\t In Parent \n");
            printf("\t\t pid = %d and ppid = %d \n\n",getpid (),getppid ());
            for (i=0;i<100;i=i+5)
            {
                    for (j=0;j<100000;j++);
                    printf("\t\t\t In Parent %d \n",i);
            }
            wait(0); /*wait for child to terminate*/
            printf("In Parent : Now the child has terminated \n");
```

Demonstrating use of fork system call


aayush@localhost:~/bookexamples - Shell - Konsole
Session Edit View Bookmarks Settings Help
1 2 3 4
XMMS - 41. Hemant Kumar -
aayush@localhost:~/bookex
5:02 pm

```
[aayush@localhost bookexamples]$ ./a.out
         In child
         pid = 2192 and ppid = 2191

         In child 0
         In child 10
         In child 20
         In child 30
         In child 40
         In child 50
         In child 60
         In child 70
         In child 80
         In child 90
                 In Parent
                 pid = 2191 and ppid = 2149

                         In Parent 0
                         In Parent 5
                         In Parent 10
                         In Parent 15
                         In Parent 20
                         In Parent 25
                         In Parent 30
                         In Parent 35
                         In Parent 40
                         In Parent 45
                         In Parent 50
                         In Parent 55
                         In Parent 60
                         In Parent 65
                         In Parent 70
                         In Parent 75
                         In Parent 80
                         In Parent 85
                         In Parent 90
                         In Parent 95
In Parent : Now the child has terminated
```

Assigning task to a newly spawned Process

```
//The Program: To get an integer
#include<stdio.h>
#include<ctype.h>
int get_integer(n_p)
    int *n_p;
{
    int c;
    int mul,sign;
    int integer_part;
    *n_p=0;
    mul=10;
    while(isspace(c = getchar())); /* skipping white space*
    if(!isdigit(c) && c!='+' && c!= '-')
    {
        /* ungetchar(c);*/
```

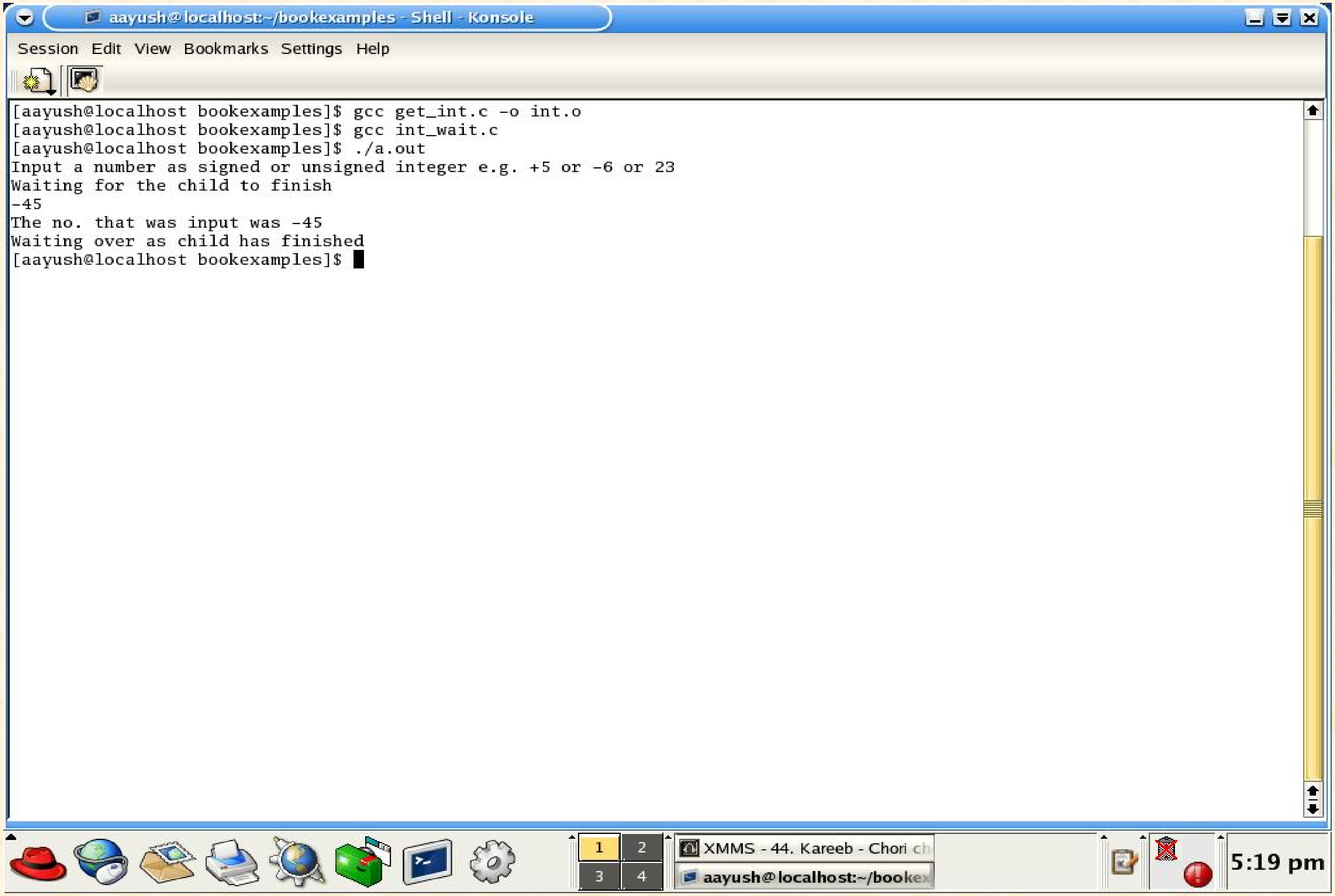
Assigning task to a newly spawned Process

```
                printf(“Found an invalid character in the integer description \n");
                return 0;
        }
        if (c=='-')sign = -1.0;
        if (c=='+') sign = 1.0;
        if (c=='-' ||c=='+') c=getchar();
        for (integer_part=0;isdigit(c);c= getchar())
        {
                integer_part=mul * integer_part +(c- '0');
};
        *n_p=integer_part;
        if(sign==-1)*n_p=-*n_p;
        if(c==EOF) return (*n_p);
}
int main()
```

Assigning task to a newly spawned Process

```
 {
 int no;
int get_integer();
printf("Input a number as signed or unsigned integer e.g. +5 or -6 or 23\n");
get_integer (&no);
printf("The no. that was input was %d \n",no);
return 0;
}
```

Assigning task to a newly spawned Process



aayush@localhost:~/bookexamples - Shell - Konsole
Session Edit View Bookmarks Settings Help
XMMS - 44. Kareeb - Chori ch
aayush@localhost:~/bookex
1 2 3 4
5:19 pm

```
[aayush@localhost bookexamples]$ gcc get_int.c -o int.o
[aayush@localhost bookexamples]$ gcc int_wait.c
[aayush@localhost bookexamples]$ ./a.out
Input a number as signed or unsigned integer e.g. +5 or -6 or 23
Waiting for the child to finish
-45
The no. that was input was -45
Waiting over as child has finished
[aayush@localhost bookexamples]$ 
```

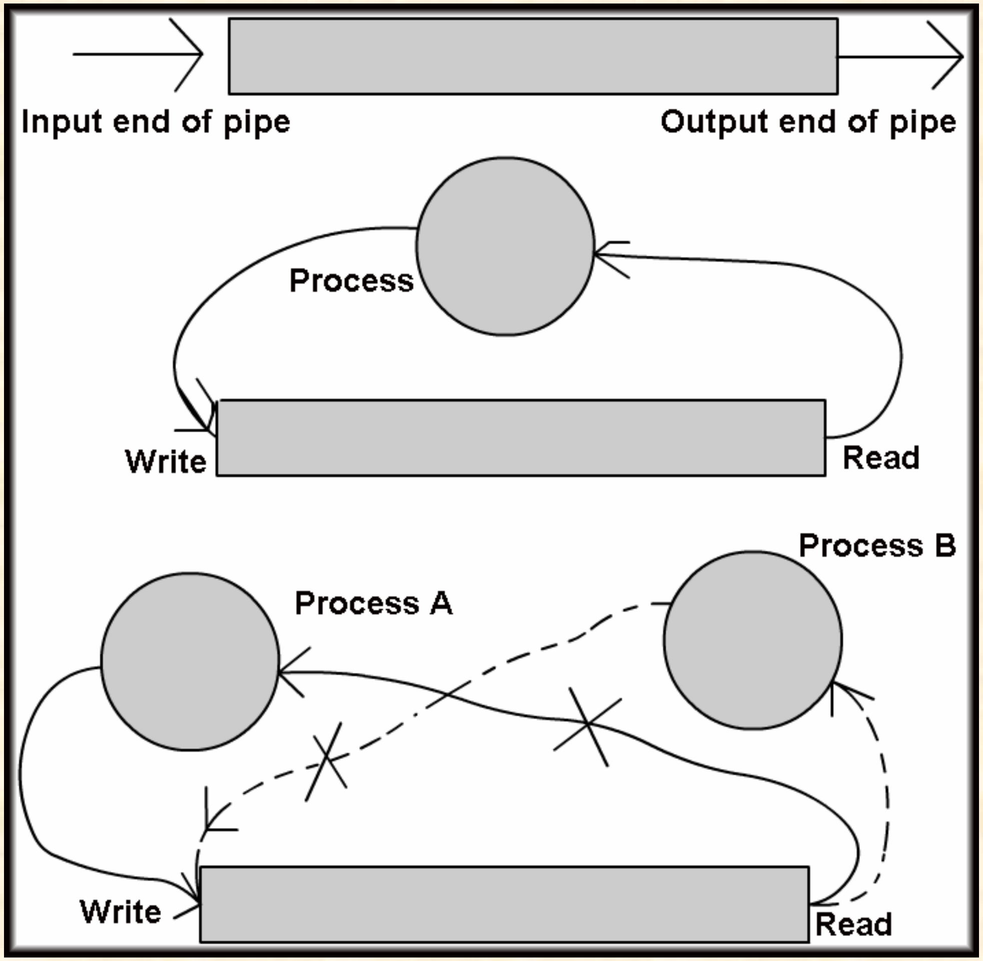
Establishing interprocess communication

- Pipes – direct the outstream of one process to feed the input of another process.
- Shared Memory location – One process write into a memory location and expect the other process to read from it.
- Message – one process sends and other interprets the message.

Pipes as Mechanism for Interprocess Communication

- The pipe is defined by *pipe(p_des).*
- The *dup* command replaces the standard I/O by pipe descriptors.
- The *execlp* command is used to populate the child process with code.
- The *close* command closes the appropriate ends of the pipe.
- The *get_str* and *rev_str* processes are pre-compiled to yield the required executables.

Pipes as IPC mechanism



Pipes as an IPC mechanism

- Unix pipes are buffers managed from within the kernel.
- A pipe operates in one direction only.
- Closing of ends is required to use a pipe.
- Pipes are not useful for processes across networks.
- Its insecure mode of communication.
- Pipes cannot support broadcast

Shared Files

- Very commonly employed IPC.
- Involves writer and reader process.
- This method does not require special system calls.
- Requires file creation, access and operations on files.
- Reader writer problem – mismatch of speed in the speed of reader and writer.

Shared Memory Communication

- Requires a certain commonly accessed area.
- Shared memory allows access to common data area even amongst the processes that are not related
- To maintain data integrity, the access is planned carefully under a user program control.

Shared Memory Model

- Set up a shared memory mechanism in the kernel.
- Identify “safe area” attach to each of the processes.
- Use shared data space in a consistent manner.
- When finished, detach the shared data space from all processes to which it is attached.
- Delete the information concerning the shared memory from the kernel.

Message-based IPC

- Very general form of communication.
- Used to send and receive formatted data streams between arbitrary processes.
- Message types helps in message interpretation.
- Usually at receiver end, messages are put in a message queue.

Signals as IPC

- One way to communicate asynchronous events.
- Signal types – generated from various sources.
- Signal handlers – offer a set of responses.

Sources of signal

- From the terminal
  - SIGINT (Ctrl C)
- From window manager
  - SIGWINCH (change in size of window)
- From other subsystems
  - SIGSEGV (external memory reference)
- From kernel
  - SIGALARM (alarm signal)
- From the processes
  - SIGKILL (kill signal)

Responses to signals

- Ignore it
- Respond to it
- Reconfigure
- Turn on/off options
- Timer information