National University of Computer and Emerging Sciences

Laboratory 2 Manual

for

Operating Systems Lab

Course Instructor	Ms. Namra Absar
Lab Instructor(s)	Rasaal Ahmad
Section	BDS-5B
Semester	Fall 2023

Department of Computer Science

FAST-NU, Lahore, Pakistan

Command Line Arguments:

Command line arguments are a way to pass data to the program. Command line arguments are passed to the main function. Suppose we want to pass two integer numbers to main function of an executable program called a.out. On the terminal write the following line:

```
./a.out 1 22
```

./a.out is the usual method of running an executable via the terminal. Here 1 and 22 are the numbers that we have passed as command line argument to the program. These arguments are passed to the main function. For the main function to be able to accept the arguments, we must change the signature of main function as follows:

```
int main(int argc, char *arg[]);
```

→ argc is the counter. It tells how many arguments have been passed.

→ arg is the character pointer to our arguments.

argc in this case will not be equal to 2, but it will be equal to 3. This is because the name ./a.out is also passed as command line argument. At index 0 of arg, we have ./a.out; at index 1, we have 1; and at index 2, we have 22. Here 1 and 22 are in the form of character string, we must convert them to integers by using a function atoi. Suppose we want to add the passed numbers and print the sum on the screen:

```
cout<< atoi(arg[1]) + atoi(arg[2]);
```

Makefiles:

- Provide a way for separate compilation.
- Describe the dependencies among the project files.
- The make utility.

Using makefiles:

Naming:

- makefile or Makefile are standard
- other name can be also used

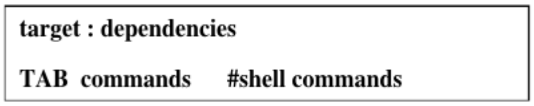
Running make:

make

make –f filename – if the name of your file is not “makefile” or “Makefile”

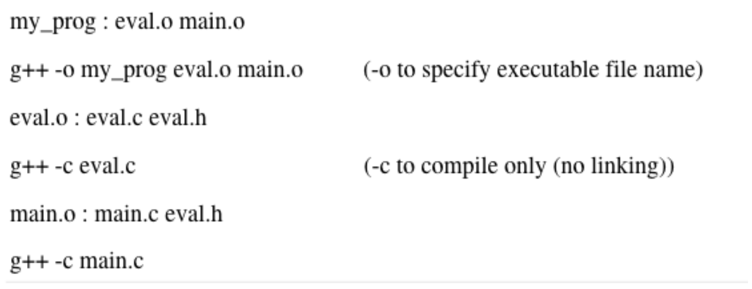
Sample makefile:

Makefiles main element is called a rule:



```
target : dependencies
TAB  commands     #shell commands
```

Example:



```
my_prog : eval.o main.o
g++ -o my_prog eval.o main.o        (-o to specify executable file name)
eval.o : eval.c eval.h
g++ -c eval.c                       (-c to compile only (no linking))
main.o : main.c eval.h
g++ -c main.c
```

<u>Topic : Fork</u>

a. What is a Process?

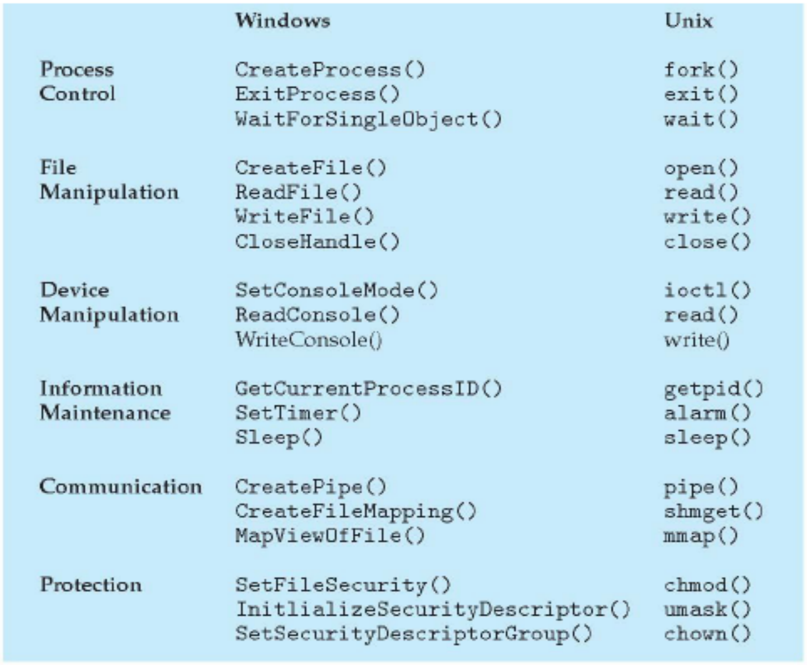
An instance of a program is called a Process. In simple terms, any command that you give to your Linux machine starts a new process.

Types of Processes:

- Foreground Processes: They run on the screen and need input from the user. For example Office Programs
- Background Processes: They run in the background and usually do not need user input. For example Antivirus.

Init is the parent of all Linux processes. It is the first process to start when a computer boots up, and it runs until the system shuts down. It is the ancestor of all other processes.

Examples of Windows and Unix System Calls



	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

1. Process Creation:

The processes in most systems can execute concurrently, and they may be created and deleted dynamically. Thus, these systems must provide a mechanism for process creation and termination.

I. fork()

- Has a return value
- Parent process => invokes fork() system call
- Continue execution from the next line after fork()
- Has its own copy of any data
- Return value is > 0 //it's the process id of the child process. This value is different from the Parents own process id.
- Child process => process created by fork() system call
- Duplicate/Copy of the parent process //LINUX
- Separate address space
- Same code segments as parent process
- Execute independently of parent process
- Continue execution from the next line right after fork()
- Has its own copy of any data
- Return value is 0

II. wait ()

- Used by the parent process
- Parent's execution is suspended
- Child remains its execution
- On termination of child, returns an exit status to the OS
- Exit status is then returned to the waiting parent process //retrieved by wait ()
- Parent process resumes execution
- #include <sys/wait.h>
- #include <sys/types.h>

III. exit()

- Process terminates its execution by calling the exit() system call
- It returns exit status, which is retrieved by the parent process using wait() command
- EXIT_SUCCESS // integer value = 0
- EXIT_FAILURE // integer value = 1
-
- OS reclaims resources allocated by the terminated process (dead process) Typically performs clean-up operations within the process space before returning control back to the OS
- _exit()
- Terminates the current process without any extra program clean-up
- Usually used by the child process to prevent from erroneously release of resources belonging to the parent process

IV. execlp() is a version of exec()

- **exec()**
- The exec family of functions replaces the current running process with a new process. It can be used to run a C program by using another C program.an executable file
- Called by an already existing process //child process

- Replaces the previous executable //overlay
- Has an exist status but cannot return anything (if exec() is successful) to the program that made the call //parent process
- Return value is -1 if not successful
- Overlay => replacement of a block of stored instructions or data with another int execlp(char const *file_path, char const *arg0,);
- Arguments beginning at arg0 are pointers to arguments to be passed to the new process.
- The first argument arg0 should be the name of the executable file.
- Example
- **execlp(/bin/ls , ls ,NULL) //lists contents of the directory**
    a. **but exec or execlp is a system call which overwrites an already existing process (calling process), so if you want to execute some code after execlp system call, then write this system call in a child process of an existing process, so it only overwrite child process.**
- Header file used -> unistd.h

2. Information Maintenance

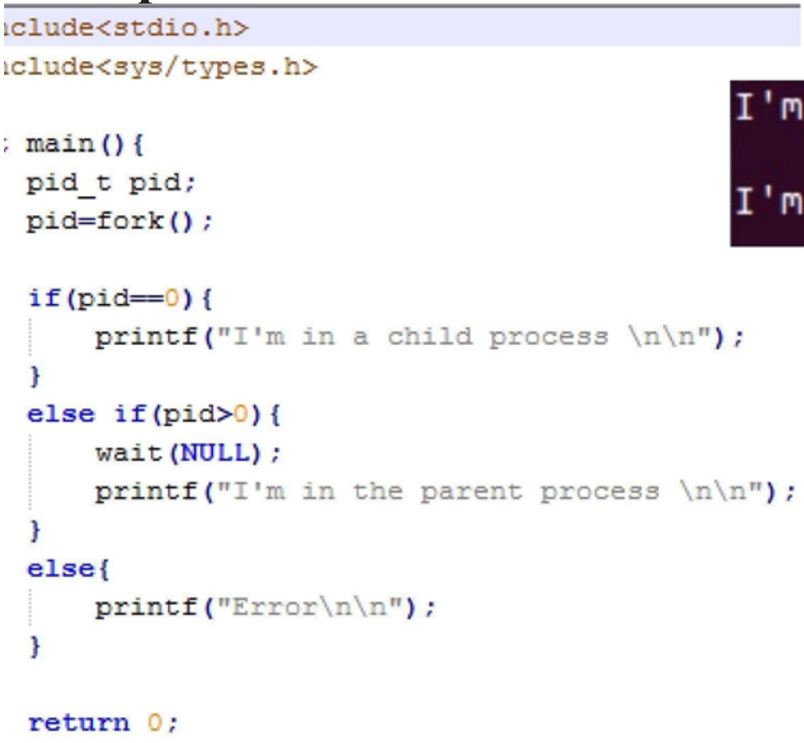
i. sleep()

- Process goes into an inactive state for a time period
- Resume execution if
- Time interval has expired
- Signal/Interrupt is received
- Takes a time value as parameter (in seconds on Unix-like OS and in milliseconds on Windows OS)
- sleep(2) // sleep for 2 seconds in Unix
- Sleep(2*1000) // sleep for 2 seconds in Windows

ii. getpid() // returns the PID of the current process

-
- getppid() // returns the PID of the parent of the current process
-
- Header files to use
-
- #include <sys/types.h>
-
- #include <unistd.h>
-
- getppid() returns 0 if the current process has no parent

Example

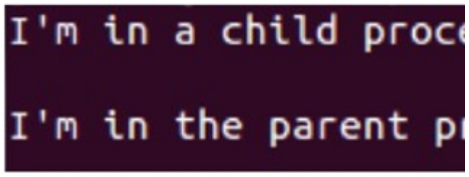


```
ıclude<stdio.h>
ıclude<sys/types.h>

; main(){
 pid_t pid;
 pid=fork();

 if(pid==0){
     printf("I'm in a child process \n\n");
 }
 else if(pid>0){
     wait(NULL);
     printf("I'm in the parent process \n\n");
 }
 else{
     printf("Error\n\n");
 }

 return 0;
```



```
I'm in a child proce

I'm in the parent pr
```

Lab Tasks

Command Line Arguments:

Question 1: Write a C or C++ program that accepts a file name as command line argument and prints the file's contents on console. If the file does not exist, print some error on the screen.

Question 2: Write a C or C++ program that accepts a list of integers as command line arguments sorts the integers and print the sorted integers on the screen.

Question 3: Write a C/C++ program that takes some integers as command line parameters, store them in an array and prints the sum and average of that array. Also note that you have to run the program for all possible error checks.

System Calls (Process Control & Information Maintenance):

Question 4: Write a C++ which performs following tasks.
Create a parent and child process using the fork command where the child process will calculate the first 15 numbers in Fibonacci series and parent process will write those numbers on screen.

Question 5: Create a program named stat that takes an integer array as command line argument (delaminated by some character such as $). The program then creates 3 child processes each of which does exactly one task from the following:

- Adds them and print the result on the screen. (Done by child 1)
- Shows average on the screen. (Done by child 2)
- Prints the maximum number on the screen. (Done by child 3)