OPERATING SYSTEM ASSIGNMENT-1 (PART-1)

WRITE-UP

Name - Rahul Khatoliya

Roll No. – 2019265

Compilation Process step by step (Command Line Options) :

1. **Preprocessing :**

   Command line argument : **$ gcc -E main.c -o main.i**

   Understanding the Meaning :

   Here **gcc** helps to invoke the GNU C compiler , **-E** enables us to stop and save the output file at preprocessed step (with an **.i extension** , meaning it's a preprocessed file) , Moreover **main.c** is our C program to be preprocessed and **-o** helps to name the resultant output file .

   Description of the Output File :

   1. It gets rid of all the comments in the source file of our program .
   2. it includes the code of the *header file(s)*, which is a file with extension .h which contains C function declarations and macro definitions .
   3. Removes all #define (which ever not used in the program) .
   4. it replaces all of the *macros* (fragments of code which have been given a name) by their values .

2. Compiling :

   Command line argument : $ gcc -S main.i -o main.s

   Understanding the Meaning :

   Here gcc helps to invoke the GNU C compiler , -S is used to stop and save the process at intermediate compilation step (with a .s extension , meaning it's an Assembly level instructions) , Moreover main.c is our C program to be and -o helps to name the resultant output file .

   Description of the Output File :

   1. File is purely based on Assembly language and contains Assembly commands .
   2. It will produce different Language designs according to different system architecture . for eg:- (x86/64) etc .
   3. Output file is the Resultant (Intermediate Representation) generated by compiler using the earlier preprocessed file .

3. Assembly :

Command line argument : **$ gcc -c main.s -o main.o**

Understanding the Meaning :

Here **gcc** helps to invoke the GNU C compiler, **-c** is used to generate object file from the earlier formed assembly .s file and itself is a machine level instructions (with a **.o extension**) , Moreover **main.c** is our C program and **-o** helps to name the resultant output file . This is the next step of compiling, in this step the **main.s** (assembly level instructions) is used as the input file and converted to the **main.o** using the Assembler which is an object file (machine level instructions).

Description of the Output File :

1. The assembler takes the IR code and transforms it into object code, that is code in machine language (i.e. binary).
2. Generated file is not human readable .
3. It has .o extension .
4. It is further linked , in the linking process .

4. Linking :

Command line argument : **$ gcc main.o -o main**

Understanding the Meaning :

Here **gcc** helps to invoke the GNU C compiler **, -o** instruct the C compiler that save the output file as main (Executable file with **.exe extension**) , Moreover **main.c** is our C program . Linking is the last stage of the compilation before producing the executable file. Linker is used to link the functions and its definitions. As linker know the address/location that where the the definitions of the function is written in the code. Also linker add some extra required code to the program like it add some standard code which is used to return the values in the program . Dynamic linking is done by the GCC by default.

Description of the Output File :

1. Can be execute using $ ./ main (executable)
2. All functions and their addresses are linked with their memories .
3. OS can easily load these types of files .
4. Some start and end code might be inserted .

SYSTEM CALLS USED IN THE PROGRAM :

1. fork() :

   Invoking syntax : fork();

   Fork system call is utilized for making another process, which is called child process, which runs simultaneously with the process that makes the **fork()** call (parent process). After another child process is made, the two processes will execute the following guidance following the **fork() system call**. A child process utilizes the equivalent pc(program counter), same CPU registers, same open documents which use in the parent process .

   It takes no parameters and returns an integer esteem. The following are various qualities returned by **fork()** :

   Negative Value : production of a child process was ineffective.

   Zero : Returned to the recently made child process.

   Positive Value : Returned to parent or caller. The worth contains process ID of recently made child process.

2. Waitpid() :

Invoking syntax :

pid_t waitpid(pid_t *pid*, int **status_ptr*, int *options*);

Suspends the calling process until a child process closes or is halted. All the more exactly, waitpid() suspends the calling process until the system gets status data on the child. On the off chance that the system as of now has status data on a suitable child when waitpid() is called, waitpid() returns right away. waitpid() is additionally finished if the calling process gets a sign whose activity is either to execute a sign handler or to end the process.

Arguments involved :

1. pid_t *pid :*
   Determines the child processes the caller needs to hang tight for: In the event that pid is more prominent than 0, waitpid() sits tight for end of the particular child whose cycle ID is equivalent to pid. In the event that pid is equivalent to zero, waitpid() hangs tight for end of any child whose cycle bunch ID is equivalent to that of the caller. On the off chance that pid is - 1, waitpid() sits tight for any child cycle to end.

   In the event that pid is not exactly - 1, waitpid() hangs tight for the end of any child whose cycle bunch ID is equivalent to the total estimation of pid.

2. int **status_ptr :*

Focuses to a location where waitpid() can store a status value. This status value is zero if the child process expressly returns zero status. Else, it is a value that can be broke down with the status analysis macros depicted in "Status Analysis Macros", beneath. The status_ptr pointer may likewise be NULL, in which case waitpid() disregards the child's bring status back.

3. int *options :*

Specifies additional information for waitpid(). The *options* value is constructed from the bitwise inclusive-OR of zero or more of the following flags defined in the sys/wait.h header file:

WCONTINUED
Reports the status of any continued child processes as well as terminated ones. The WIFCONTINUED macro lets a process distinguish between a continued process and a terminated one.

WNOHANG
Demands status information immediately. If status information is immediately available on an appropriate child process, waitpid() returns this information. Otherwise, waitpid() returns immediately with an error code indicating that the information was not available. In other words, WNOHANG checks child processes without causing the caller to be suspended.

WUNTRACED
Reports on stopped child processes as well as terminated ones. The WIFSTOPPED macro lets a process distinguish between a stopped process and a terminated one.

3. Exit() :

Invoking syntax : void exit (int status);

exit() ends the process typically.

status: Status esteem got back to the parent process. For the most part, a status estimation of 0 or EXIT_SUCCESS demonstrates success, and some other value or the steady EXIT_FAILURE is utilized to show a blunder. exit() performs following activities.

* Flushes unwritten cushioned information.

* Closes every open record.

* Removes impermanent records.

* Returns a whole number exit status to the working system.

We can see that the argument passed here is the status , i.e 0 or EXIT_SUCCESS and some other value or the steady EXIT_FAILURE .

Working Of the Program :

Header files used :

<sys/wait.h> , <stdlib.h>, <unistd.h>, <stdio.h> and <string.h>

Program Explanation :

We have to create processes using fork() system call , in which we were asked to Calculate Average Score of the various assignments provided and print the details of the student , but we were asked to perform it separately section wise in child and parent process respectively . Hence we were playing around child's pid which will ensure that process is going on I which state , and later provide information for parent to begin its execution after waiting for the child's process to end.

Moreover we have used waitpid for the sake of waiting for the child's process to end , later executing the parent . The data was provided from the .csv file , so in order to fetch the data we have used stdio library to read the same file , and print the stuffs according to the respective processes. Also , to move from child's process to parent's , we had used the exit system call , which will terminate the child's process and sends some instruction to the parent's one , which in turn begin to make its execution .

In order to separate the details section wise , we have compared the value of the section entity , and if it belongs to the child , then those student will get reflected on child's process , and if not , then will be printed on parent's process . Basically , we are reading the file twice , once in child's process and secondly on parent's side , so as to reflect the desired changes .

In addition , we had taken care of some possible errors or exceptions , which might take place due to some unavoidable circumstances , which will be described in the next page .

Errors Handled :

1. Due to some discrepancy , it might happen that production of child process was ineffective , and due to which the fork() will return -1 , indicating the same mishappening , Thus in order to print those error details we have used perror which would print the corresponding message in the stderr .

2. The file provided might be faulty in some cases , thus we need to ensure that the same if we are reading is not NULL , i.e not empty , and that's why we have taken care for such circumstances , by simply priniting the message if such case found .

3. Now suppose if due to some issues , the returned value by waitpid was -1 , that means , we hadn't successfully achieved are goal , to collect the information of the just finished/running child's process , thus we need to handle such incidences , therefore we have printed the same , in the stderr with the help of perror .