

**LAPORAN PRAKTIKUM
SISTEM OPERASI
MODUL 8**



DISUSUN OLEH:

NIM	L200220277
NAMA	MHD. FARHAN LUBIS
KELAS	F

**PROGRAM STUDI INFORMATIKA
FAKULTAS KOMUNIKASI DAN INFORMATIKA
UNIVERSITAS MUHAMMADIYAH SURAKARTA**

2023

DAFTAR ISI

DAFTAR ISI.....	2
LANGKAH KERJA	5
1. Membuat sebuah ‘child process’ (proses baru) dengan menggunakan system call ‘fork’.....	5
a. Deklarasi sebuah variabel x yang akan diakses bersama antara child proses dan parent proses.	6
b. Membuat sebuah child proses menggunakan system call fork.....	6
c. Jika return value bernilai -1, tampilkan teks ‘Pembuatan proses GAGAL’, dilanjutkan dengan keluar program dengan perintah system call ‘exit’.	6
d. Jika return value sama dengan 0 (NOL), Tampilkan teks ‘Child Process’, tampilkan ID proses dari child proses menggunakan perintah system call ‘getpid’, tampilkan nilai x, dan tampilkan ID proses parent dengan perintah system call ‘getppid’.....	6
e. Untuk nilai return value yang lainnya, tampilkan teks ‘Parent process’, tampilkan ID dari parent proses menggunakan perintah system call getpid, tampilkan nilai x, dan tampilkan ID dari proses shell menggunakan perintah system call getppid.	6
f. Stop	6
Kode program:.....	6
Terminal ketika program dicompile & dijalankan:	7
2. Menghentikan sementara (block) proses parent sampai dengan proses child selesai, menggunakan perintah system call ‘wait’.	7
a. Membuat sebuah child proses menggunakan sytem call ‘fork’.....	8
b. Jika return value bernilai -1, selanjutnya tampilkan teks ‘pembuatan proses gagal’, dan keluar program dengan menggunakan perintah system call ‘exit’.	8
c. Jika return value berupa angka positif (> 0), ‘pause’ hentikan sementara ‘parent’ proses tunggu sampai child proses berakhir dengan menggunakan perintah system call ‘wait’. Tampilkan teks ‘Parent starts’, selanjutnya tampilkan nomor genap mulai dari 0 sd 10, terakhir tampilkan teks ‘Parent end’.	8
d. Jika return value bernilai 0 (NOL), tampilkan teks ‘Child start’, tampilkan nomor ganjil mulai dari 0 s/d 10, selanjutnya tampilkan teks ‘child ends’.....	8
e. Stop	8
Kode program:.....	8
Terminal ketika program dicompile & dijalankan:	10

3. Loading program yang dapat dieksekusi dalam sebuah ‘child’ proses menggunakan perintah system call ‘exec’. Membuat program dengan algoritma sebagai berikut:	10
a. Jika terdapat 3 argumen dalam command-line berhenti (stop).	11
b. Membuat child proses dengan perintah system call ‘fork’	11
c. Jika return value adalah -1, selanjutnya tampilkan teks ‘Pembuatan proses Gagal’, dan keluar program dengan perintah system call exit.	11
d. Jika return value >0 (positif), selanjutnya hentikan parent-proses sementara hingga child-proses berakhir dengan menggunakan perintah system call wait. Tampilkan teks ‘Child berakhir’, dan hentikan parent-proses.	11
e. Jika return value sama dengan 0 (NOL), selanjutnya tampilkan teks ‘Child starts’, load program dari lokasi yang diberikan dalam ‘path’ ke dalam child-proses, menggunakan perintah system call ‘exec’. Jika return value dari perintah ‘exec’ adalah bilangan negatif, tampilkan error yang terjadi dan stop. Hentikan child- proses.	11
f. Stop	11
Kode program:.....	11
Terminal ketika program dicompile & dijalankan:	13
4. Menampilkan status file menggunakan perintah system call ‘stat’	13
a. Gunakan ‘nama file’ yang diberikan melalui argumen dalam perintah command-line.14	
b. Jika ‘nama-file’ tidak ada maka stop disini (keluar program)	14
c. Panggil system call ‘stat’ pada ‘nama-file’ tersebut yang akan mengembalikan sebuah struktur.....	14
d. Tampilkan informasi mengenai st_uid, st_blksize, st_block, st_size, st_nlink, etc. 14	
e. Ubah waktu dalam st_time, st_mtime dengan menggunakan fungsi ctime.	14
f. Bandingkan st_mode dengan konstanta mode seperti S_IRUSR, S_IWGRP, S_IXOTH dan tampilkan informasi mengenai ‘file-permissions’.	14
g. Stop	14
Kode program:.....	14
Terminal ketika program dicompile & dijalankan:	17
5. Menampilkan isi direktori menggunakan perintah system call ‘readdir’	18
a. Gunakan ‘nama-direktori’ yang diberikan sebagai argumen pada command-line. 19	
b. Jika direktori tidak ditemukan stop, keluar program.....	19
c. Buka direktori menggunakan perintah system call ‘opendir’ yang akan menghasilkan sebuah struktur.	19

d. Baca direktori menggunakan perintah system call 'readdir' yang juga akan menghasilkan struktur data.	19
e. Tampilkan d_name (nama direktori)	19
f. Akhiri pembacaan direktori dengan perintah system call 'closedir'.	19
g. Stop	19
Kode program:	19
Terminal ketika program dicompile & dijalankan:	20

LANGKAH KERJA

Gunakan aplikasi ‘nano’ atau ‘vi’ atau teks editor yang lain untuk mengedit kode program berikut, Selanjutnya untuk melakukan kompilasi dapat dilakukan dengan perintah berikut:

```
$gcc 'nama_file.c'
```

Jika tidak ada kesalahan maka akan dihasilkan sebuah program bernama ‘a.out’, dan untuk menjalankan program tersebut dapat dilakukan dengan cara berikut:

```
$ ./a.out
```

Jika pada PC anda tidak tersedia compiler ‘gcc’ dapat digunakan fasilitas online compiler yang disediakan oleh link berikut:

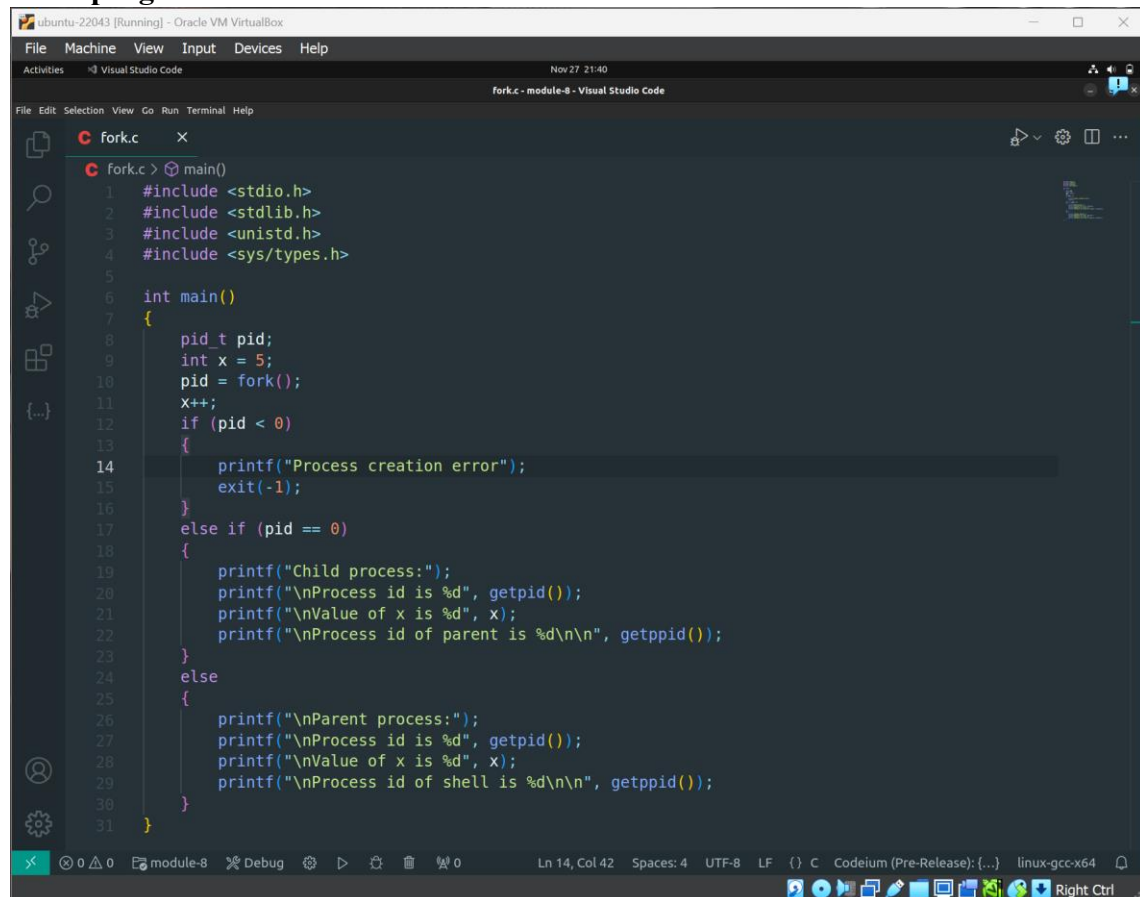
https://www.tutorialspoint.com/compile_c_online.php

1. Membuat sebuah ‘child process’ (proses baru) dengan menggunakan system call ‘fork’.

Membuat program dengan algoritma sebagai berikut: (contoh program diberikan pada bagian berikutnya).

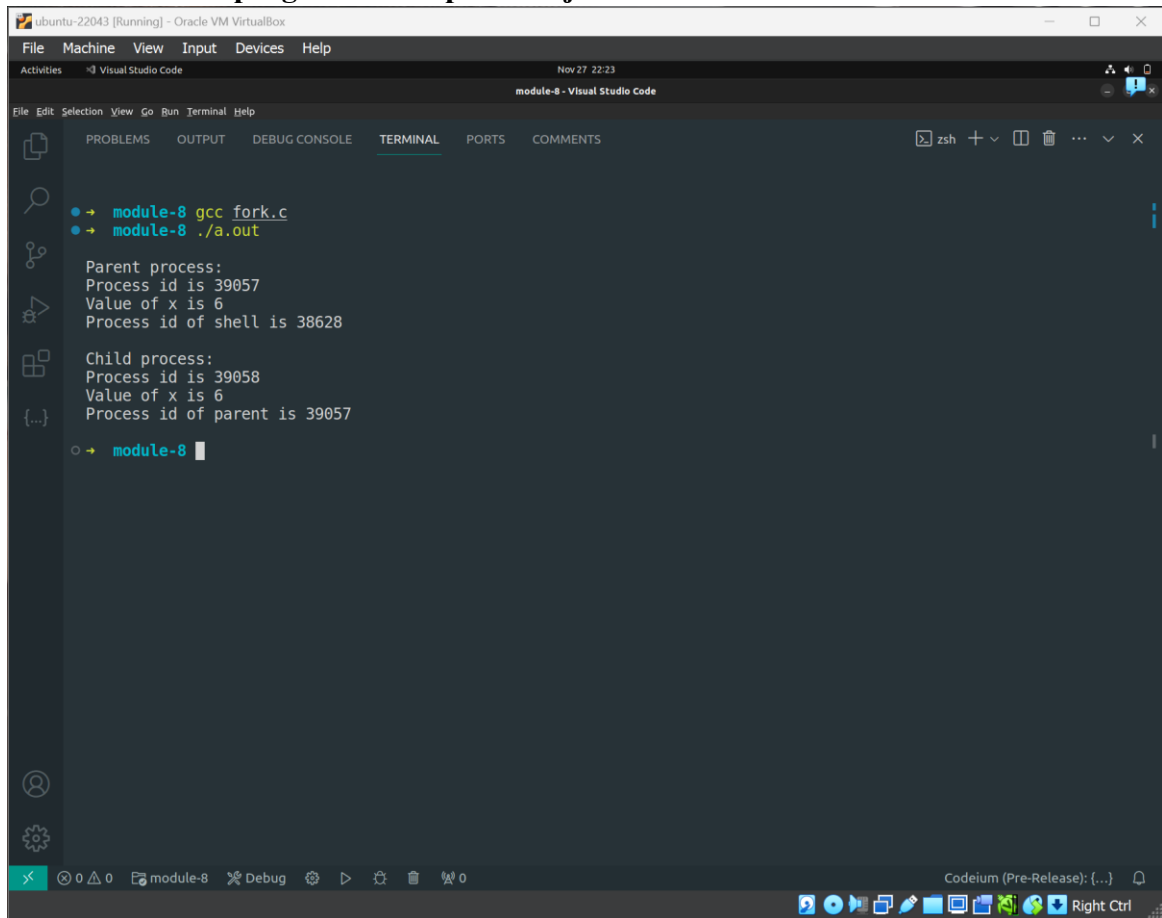
- a. Deklarasi sebuah variabel x yang akan diakses bersama antara child proses dan parent proses.
- b. Membuat sebuah child proses menggunakan system call fork.
- c. Jika return value bernilai -1, tampilkan teks 'Pembuatan proses GAGAL', dilanjutkan dengan keluar program dengan perintah system call 'exit'.
- d. Jika return value sama dengan 0 (NOL), Tampilkan teks 'Child Process', tampilkan ID proses dari child proses menggunakan perintah system call 'getpid', tampilkan nilai x, dan tampilkan ID proses parent dengan perintah system call 'getppid'.
- e. Untuk nilai return value yang lainnya, tampilkan teks 'Parent process', tampilkan ID dari parent proses menggunakan perintah system call getpid, tampilkan nilai x, dan tampilkan ID dari proses shell menggunakan perintah system call getppid.
- f. Stop

Code program:



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5
6 int main()
7 {
8     pid_t pid;
9     int x = 5;
10    pid = fork();
11    x++;
12    if (pid < 0)
13    {
14        printf("Process creation error");
15        exit(-1);
16    }
17    else if (pid == 0)
18    {
19        printf("Child process:");
20        printf("\nProcess id is %d", getpid());
21        printf("\nValue of x is %d", x);
22        printf("\nProcess id of parent is %d\n", getppid());
23    }
24    else
25    {
26        printf("\nParent process:");
27        printf("\nProcess id is %d", getpid());
28        printf("\nValue of x is %d", x);
29        printf("\nProcess id of shell is %d\n", getppid());
30    }
31 }
```

Terminal ketika program dicompile & dijalankan:



```
module-8 gcc fork.c
module-8 ./a.out

Parent process:
Process id is 39057
Value of x is 6
Process id of shell is 38628

Child process:
Process id is 39058
Value of x is 6
Process id of parent is 39057

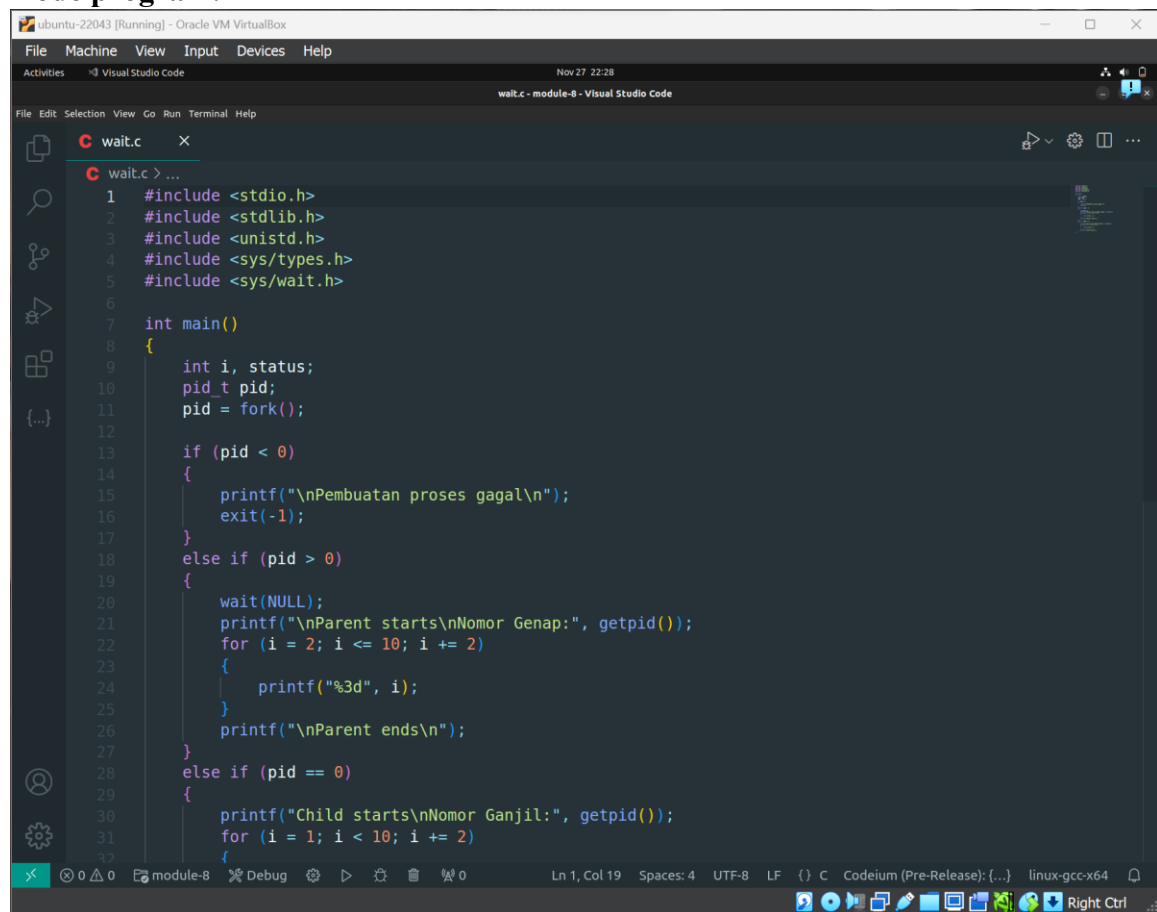
module-8
```

2. Menghentikan sementara (block) proses parent sampai dengan proses child selesai, menggunakan perintah system call 'wait'.

Membuat program dengan algoritma sebagai berikut, contoh program diberikan pada bagian berikutnya.

- a. Membuat sebuah child proses menggunakan sytem call 'fork'.
- b. Jika return value bernilai -1, selanjutnya tampilkan teks 'pembuatan proses gagal', dan keluar program dengan menggunakan perintah system call 'exit'.
- c. Jika return value berupa angka positif (> 0), 'pause' hentikan sementara 'parent' proses tunggu sampai child proses berakhir dengan menggunakan perintah system call 'wait'. Tampilkan teks 'Parent starts', selanjutnya tampilkan nomor genap mulai dari 0 s/d 10, terakhir tampilkan teks 'Parent end'.
- d. Jika return value bernilai 0 (NOL), tampilkan teks 'Child start', tampilkan nomor ganjil mulai dari 0 s/d 10, selanjutnya tampilkan teks 'child ends'.
- e. Stop

Kode program:

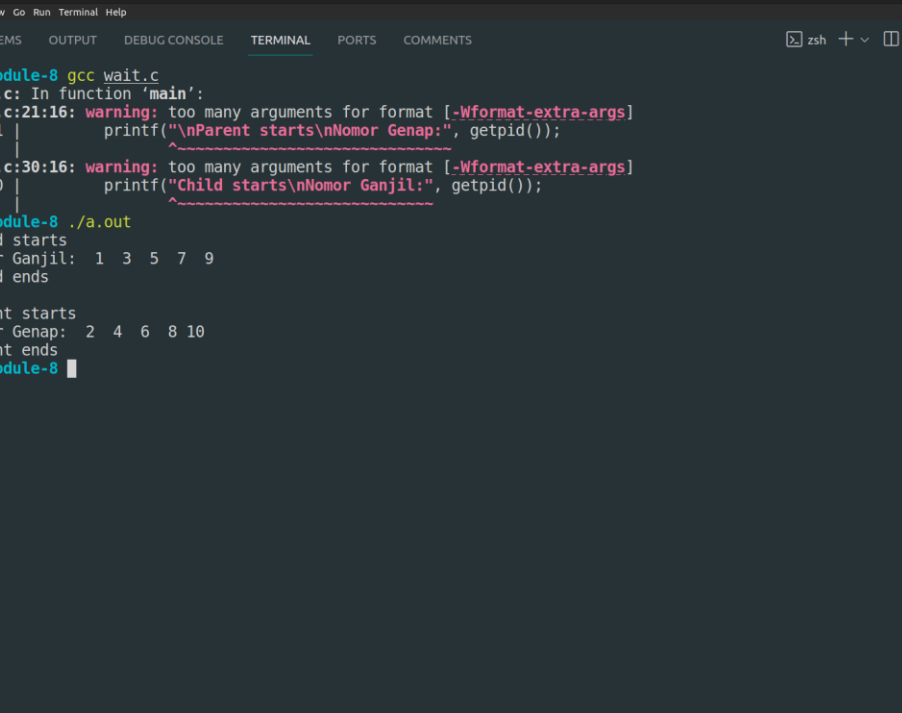


```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6
7 int main()
8 {
9     int i, status;
10    pid_t pid;
11    pid = fork();
12
13    if (pid < 0)
14    {
15        printf("\nPembuatan proses gagal\n");
16        exit(-1);
17    }
18    else if (pid > 0)
19    {
20        wait(NULL);
21        printf("\nParent starts\nNomor Genap:", getpid());
22        for (i = 2; i <= 10; i += 2)
23        {
24            printf("%3d", i);
25        }
26        printf("\nParent ends\n");
27    }
28    else if (pid == 0)
29    {
30        printf("Child starts\nNomor Ganjil:", getpid());
31        for (i = 1; i < 10; i += 2)
32        {
```



```
wait.c > ...
6
7  int main()
8  {
9      int i, status;
10     pid_t pid;
11     pid = fork();
12
13     if (pid < 0)
14     {
15         printf("\nPembuatan proses gagal\n");
16         exit(-1);
17     }
18     else if (pid > 0)
19     {
20         wait(NULL);
21         printf("\nParent starts\nNomor Genap:", getpid());
22         for (i = 2; i <= 10; i += 2)
23         {
24             printf("%3d", i);
25         }
26         printf("\nParent ends\n");
27     }
28     else if (pid == 0)
29     {
30         printf("Child starts\nNomor Ganjil:", getpid());
31         for (i = 1; i < 10; i += 2)
32         {
33             printf("%3d", i);
34         }
35         printf("\nChild ends\n");
36     }
37 }
```

Terminal ketika program dicompile & dijalankan:



The screenshot shows a Visual Studio Code editor window with a terminal pane at the bottom. The terminal is running a C program named `wait.c` in a directory named `module-8`. The program uses `fork()` to create a child process. The parent process prints "Parent starts" and "Parent ends". The child process prints "Child starts" and "Child ends". The terminal output shows the program running successfully with some warnings about format strings.

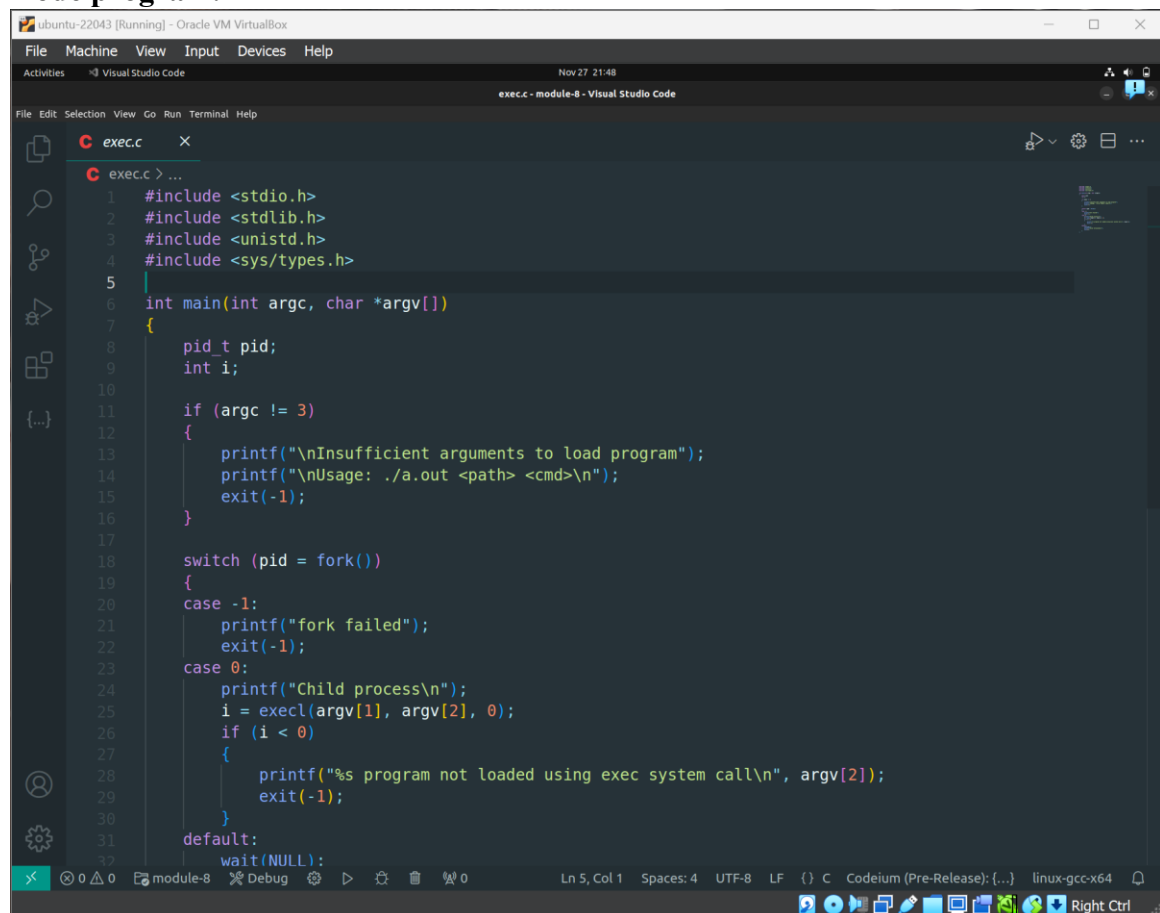
```
→ module-8 gcc wait.c
wait.c: In function 'main':
wait.c:21:16: warning: too many arguments for format [-Wformat-extra-args]
   21 |     printf("\nParent starts\nNomor Genap:", getpid());
      |           ^~
wait.c:30:16: warning: too many arguments for format [-Wformat-extra-args]
   30 |     printf("Child starts\nNomor Ganjil:", getpid());
      |           ^~
→ module-8 ./a.out
Child starts
Nomor Ganjil: 1 3 5 7 9
Child ends

Parent starts
Nomor Genap: 2 4 6 8 10
Parent ends
→ module-8
```

- 3. Loading program yang dapat dieksekusi dalam sebuah ‘child’ proses menggunakan perintah system call ‘exec’. Membuat program dengan algoritma sebagai berikut:**
(contoh program diberikan pada bagian berikutnya).

- a. Jika terdapat 3 argumen dalam command-line berhenti (stop).
- b. Membuat child proses dengan perintah system call 'fork'
- c. Jika return value adalah -1, selanjutnya tampilkan teks 'Pembuatan proses Gagal', dan keluar program dengan perintah system call exit.
- d. Jika return value >0 (positif), selanjutnya hentikan parent-proses sementara hingga child-proses berakhir dengan menggunakan perintah system call wait. Tampilkan teks 'Child berakhir', dan hentikan parent-proses.
- e. Jika return value sama dengan 0 (NOL), selanjutnya tampilkan teks 'Child starts', load program dari lokasi yang diberikan dalam 'path' ke dalam child-proses, menggunakan perintah system call 'exec'. Jika return value dari perintah 'exec' adalah bilangan negatif, tampilkan error yang terjadi dan stop. Hentikan child- proses.
- f. Stop

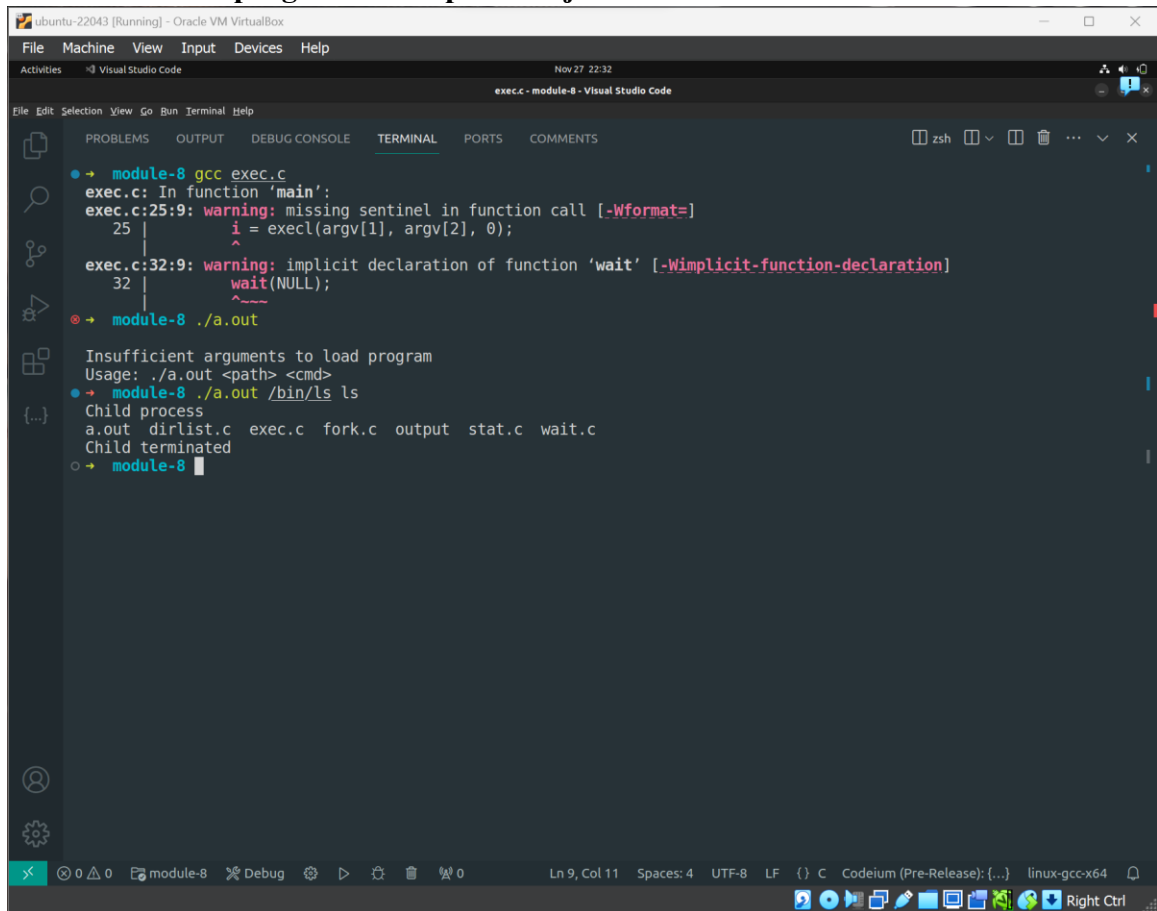
Kode program:



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5
6 int main(int argc, char *argv[])
7 {
8     pid_t pid;
9     int i;
10
11     if (argc != 3)
12     {
13         printf("\nInsufficient arguments to load program");
14         printf("\nUsage: ./a.out <path> <cmd>\n");
15         exit(-1);
16     }
17
18     switch (pid = fork())
19     {
20     case -1:
21         printf("fork failed");
22         exit(-1);
23     case 0:
24         printf("Child process\n");
25         i = execl(argv[1], argv[2], 0);
26         if (i < 0)
27         {
28             printf("%s program not loaded using exec system call\n", argv[2]);
29             exit(-1);
30         }
31     default:
32         wait(NULL);
33     }
```

```
exec.c > ...
6  int main(int argc, char *argv[])
7  {
8      pid_t pid;
9      int i;
10
11     if (argc != 3)
12     {
13         printf("\nInsufficient arguments to load program");
14         printf("\nUsage: ./a.out <path> <cmd>\n");
15         exit(-1);
16     }
17
18     switch (pid = fork())
19     {
20     case -1:
21         printf("fork failed");
22         exit(-1);
23     case 0:
24         printf("Child process\n");
25         i = execl(argv[1], argv[2], 0);
26         if (i < 0)
27         {
28             printf("%s program not loaded using exec system call\n", argv[2]);
29             exit(-1);
30         }
31     default:
32         wait(NULL);
33         printf("Child terminated\n");
34         exit(0);
35     }
36 }
```

Terminal ketika program dicompile & dijalankan:



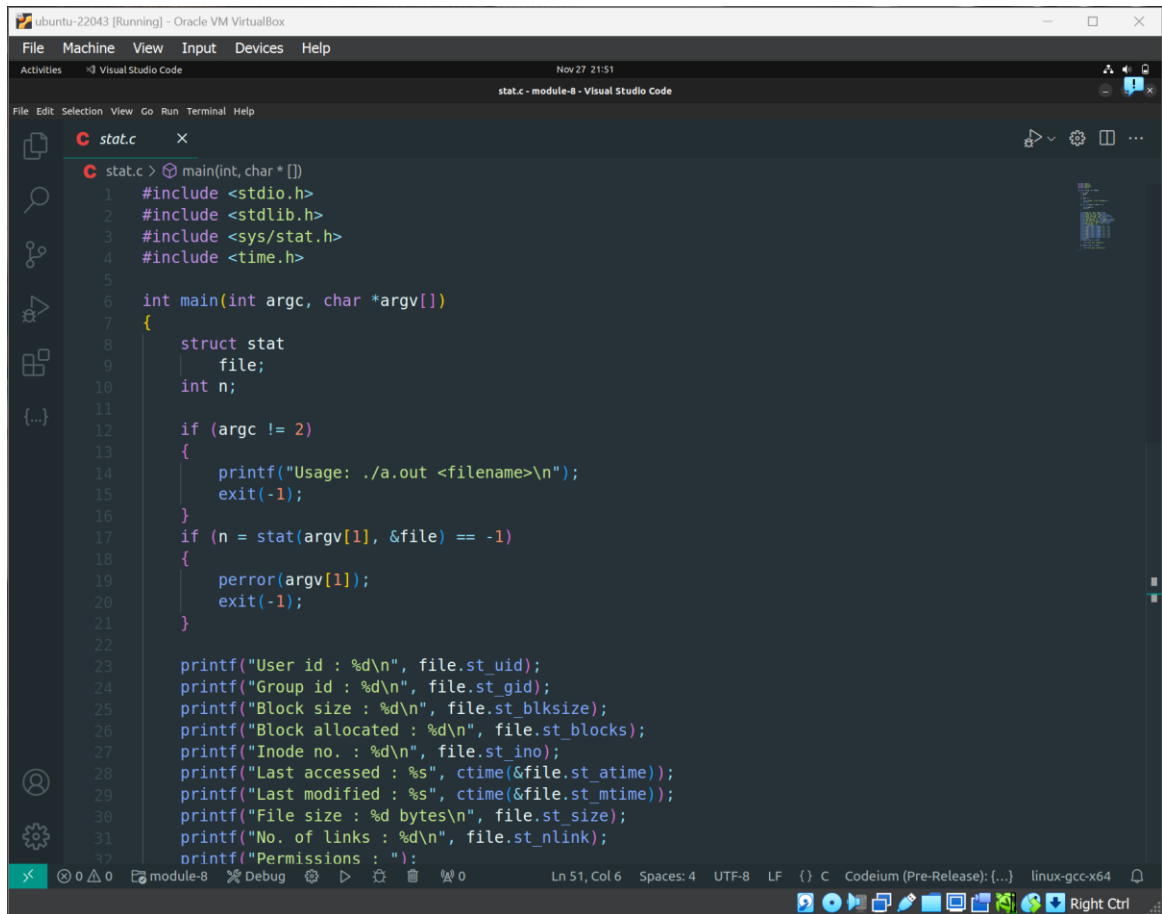
```
module-8 gcc exec.c
exec.c: In function 'main':
exec.c:25:9: warning: missing sentinel in function call [-Wformat=]
    25 |         i = execl(argv[1], argv[2], 0);
        |         ^
exec.c:32:9: warning: implicit declaration of function 'wait' [-Wimplicit-function-declaration]
    32 |         wait(NULL);
        |         ^
module-8 ./a.out
Insufficient arguments to load program
Usage: ./a.out <path> <cmd>
module-8 ./a.out /bin/ls ls
Child process
a.out dirlist.c exec.c fork.c output stat.c wait.c
Child terminated
module-8
```

4. Menampilkan status file menggunakan perintah system call 'stat'

Membuat program dengan algoritma sebagai berikut (contoh code ada di bagian berikutnya):

- a. Gunakan 'nama file' yang diberikan melalui argumen dalam perintah command-line.
- b. Jika 'nama-file' tidak ada maka stop disini (keluar program)
- c. Panggil system call 'stat' pada 'nama-file' tersebut yang akan mengembalikan sebuah struktur
- d. Tampilkan informasi mengenai st_uid, st_blksize, st_block, st_size, st_nlink, etc.
- e. Ubah waktu dalam st_time, st_mtime dengan menggunakan fungsi ctime.
- f. Bandingkan st_mode dengan konstanta mode seperti S_IRUSR, S_IWGRP, S_IXOTH dan tampilkan informasi mengenai 'file-permissions'.
- g. Stop

Kode program:



```
stat.c > main(int, char * [])
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/stat.h>
4 #include <time.h>
5
6 int main(int argc, char *argv[])
7 {
8     struct stat
9     file;
10     int n;
11
12     if (argc != 2)
13     {
14         printf("Usage: ./a.out <filename>\n");
15         exit(-1);
16     }
17     if (n = stat(argv[1], &file) == -1)
18     {
19         perror(argv[1]);
20         exit(-1);
21     }
22
23     printf("User id : %d\n", file.st_uid);
24     printf("Group id : %d\n", file.st_gid);
25     printf("Block size : %d\n", file.st_blksize);
26     printf("Block allocated : %d\n", file.st_blocks);
27     printf("Inode no. : %d\n", file.st_ino);
28     printf("Last accessed : %s", ctime(&file.st_atime));
29     printf("Last modified : %s", ctime(&file.st_mtime));
30     printf("File size : %d bytes\n", file.st_size);
31     printf("No. of links : %d\n", file.st_nlink);
32     printf("Permissions : "):
```

```
32 printf("Permissions : ");
33 printf((S_ISDIR(file.st_mode)) ? "d" : "-");
34 printf((file.st_mode & S_IRUSR) ? "r" : "-");
35 printf((file.st_mode & S_IWUSR) ? "w" : "-");
36 printf((file.st_mode & S_IXUSR) ? "x" : "-");
37 printf((file.st_mode & S_IRGRP) ? "r" : "-");
38 printf((file.st_mode & S_IWGRP) ? "w" : "-");
39 printf((file.st_mode & S_IXGRP) ? "x" : "-");
40 printf((file.st_mode & S_IROTH) ? "r" : "-");
41 printf((file.st_mode & S_IWOTH) ? "w" : "-");
42 printf((file.st_mode & S_IXOTH) ? "x" : "-");
43 printf("\n");
44 if (file.st_mode & __S_IFREG)
45 {
46     printf("File type : Regular\n");
47 }
48 if (file.st_mode & __S_IFDIR)
49 {
50     printf("File type : Directory\n");
51 }
52 }
```


Terminal ketika program dcompile & dijalankan:

[illegible]

The screenshot shows a Visual Studio Code editor window titled 'exec.c - module-8 - Visual Studio Code'. The editor displays a C program with three printf statements, each with a warning about format specifiers. The warnings are: 'format '%d' expects argument of type 'int', but argument 2 has type '__ino_t' {aka 'long unsigned int'} [-Wformat=]', 'format '%d' expects argument of type 'int', but argument 2 has type '__off_t' {aka 'long int'} [-Wformat=]', and 'format '%d' expects argument of type 'int', but argument 2 has type '__nlink_t' {aka 'long unsigned int'} [-Wformat=]'. The code defines variables of type int and casts them to __ino_t, __off_t, and __nlink_t. The terminal output shows the execution of ./a.out stat.c, displaying file statistics for stat.c.

```
int          __blkcnt_t {aka long int}
%ld

stat.c:27:26: warning: format '%d' expects argument of type 'int', but argument 2 has type '__ino_t' {aka 'long unsigned int'} [-Wformat=]
27      printf("Inode no. : %d\n", file.st_ino);
                                ^~
                                |
                                |__ino_t {aka long unsigned int}
                                %ld

stat.c:30:26: warning: format '%d' expects argument of type 'int', but argument 2 has type '__off_t' {aka 'long int'} [-Wformat=]
30      printf("File size : %d bytes\n", file.st_size);
                                ^~
                                |
                                |__off_t {aka long int}
                                %ld

stat.c:31:29: warning: format '%d' expects argument of type 'int', but argument 2 has type '__nlink_t' {aka 'long unsigned int'} [-Wformat=]
31      printf("No. of links : %d\n", file.st_nlink);
                                ^~
                                |
                                |__nlink_t {aka long unsigned int}
                                %ld

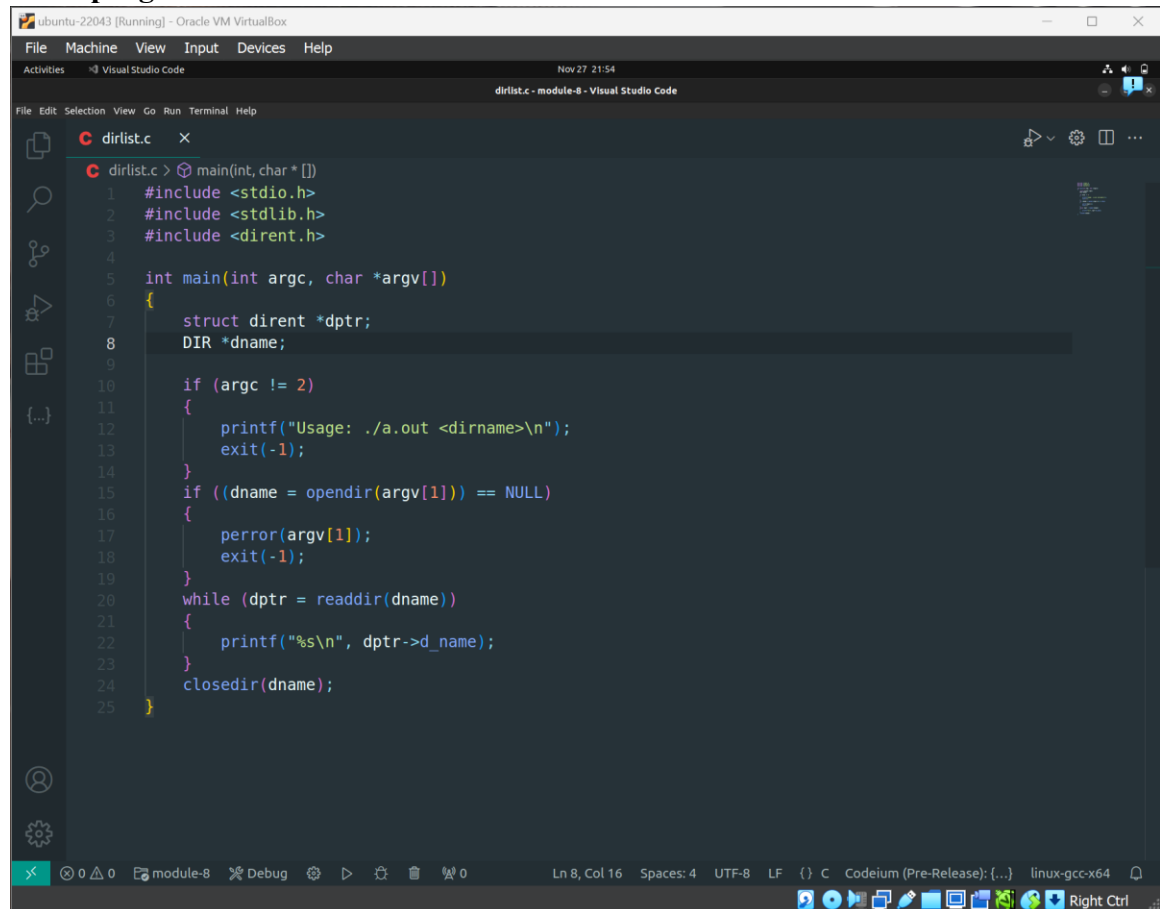
➤ module-8 ./a.out
Usage: ./a.out <filename>
➤ module-8 ./a.out stat.c
User id : 1000
Group id : 1000
Block size : 4096
Block allocated : 8
Inode no. : 1218458
Last accessed : Mon Nov 27 20:46:10 2023
Last modified : Tue Nov 21 10:36:42 2023
File size : 1525 bytes
No. of links : 1
Permissions : -rw-rw-r--
File type : Regular
➤ module-8
```

5. Menampilkan isi direktori menggunakan perintah system call 'readdir'

Membuat program dengan algoritma sebagai berikut (contoh code ada di bagian berikutnya):

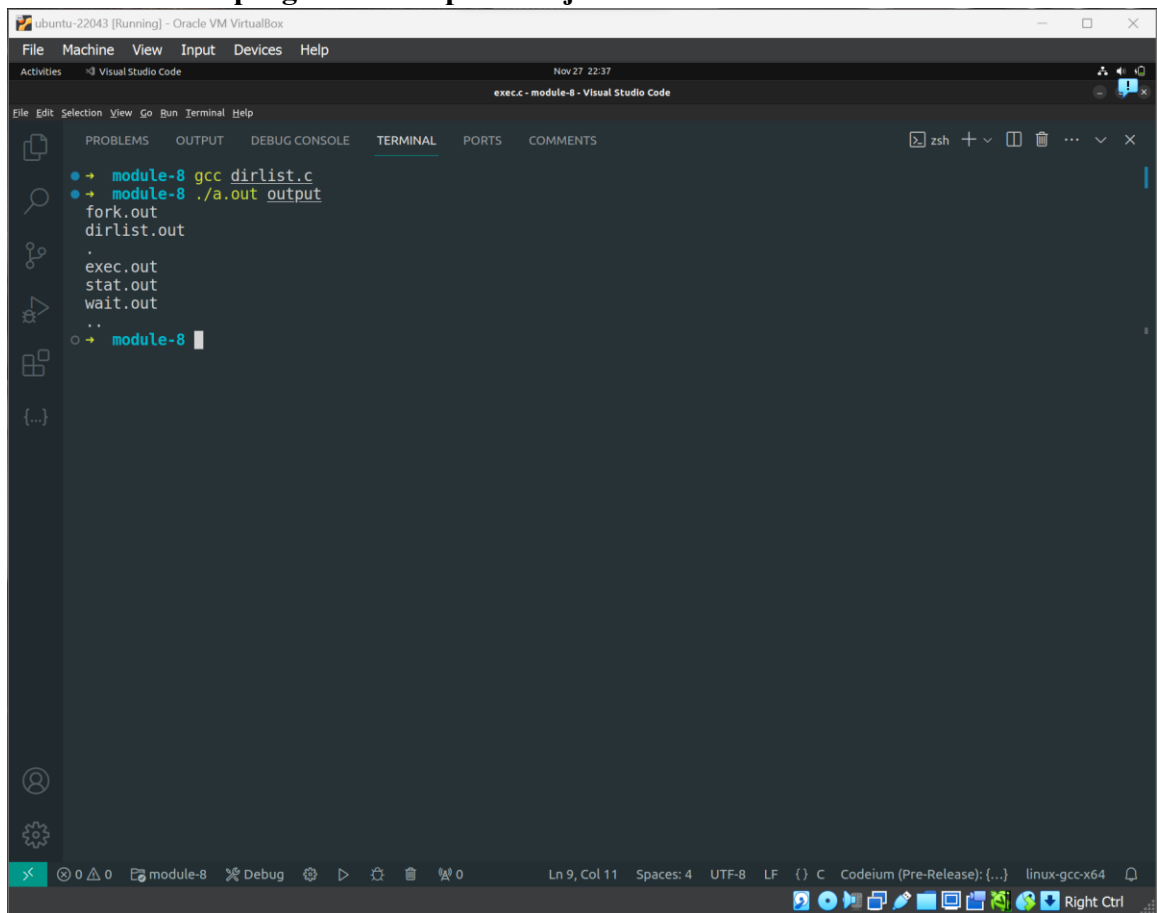
- a. Gunakan 'nama-direktori' yang diberikan sebagai argumen pada command-line.
- b. Jika direktori tidak ditemukan stop, keluar program
- c. Buka direktori menggunakan perintah system call 'opendir' yang akan menghasilkan sebuah struktur.
- d. Baca direktori menggunakan perintah system call 'readdir' yang juga akan menghasilkan struktur data.
- e. Tampilkan d_name (nama direktori)
- f. Akhiri pembacaan direktori dengan perintah system call 'closedir'.
- g. Stop

Kode program:



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <dirent.h>
4
5 int main(int argc, char *argv[])
6 {
7     struct dirent *dptr;
8     DIR *dname;
9
10    if (argc != 2)
11    {
12        printf("Usage: ./a.out <dirname>\n");
13        exit(-1);
14    }
15    if ((dname = opendir(argv[1])) == NULL)
16    {
17        perror(argv[1]);
18        exit(-1);
19    }
20    while (dptr = readdir(dname))
21    {
22        printf("%s\n", dptr->d_name);
23    }
24    closedir(dname);
25 }
```

Terminal ketika program dicompile & dijalankan:



The screenshot shows a terminal window within a Visual Studio Code editor. The terminal output displays the compilation and execution of a program. The commands entered are `gcc dirlist.c` and `./a.out output`. The output shows the files `fork.out`, `dirlist.out`, `exec.out`, `stat.out`, and `wait.out`. The terminal window is titled "exec.c - module-8 - Visual Studio Code" and the status bar at the bottom indicates the current file is `module-8` and the compiler is `linux-gcc-x64`.

```
module-8 gcc dirlist.c
module-8 ./a.out output
fork.out
dirlist.out
.
exec.out
stat.out
wait.out
..
module-8
```