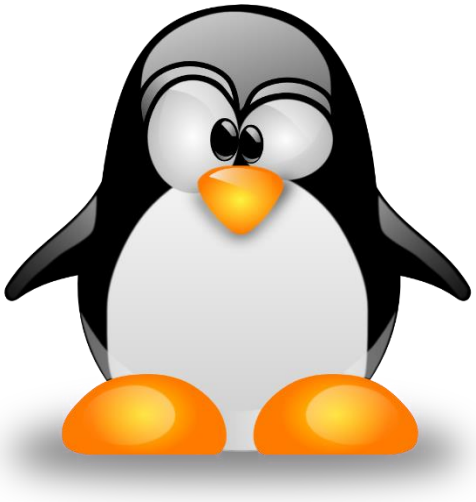
Laporan Praktikum 2
Sistem Operasi (E)



Kelompok E14:
Rani Aulia H 5114100044
Luqman Ahmad 5114100187

1. Membuat Shell
2. Mencari Jumlah Bilangan Prima Kurang dari N
3. Membuat Aplikasi Multithread untuk Menyalin Isi File

Membuat Shell

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<signal.h>

#define COLOR_RED "\33[0:31m\\]"
#define COLOR_END "\33[0m\\]"
#define BUFFSIZE 1024
#define DELIMITER " \t\r\n\a"
```

Di awal, definisikan *BUFFSIZE* adalah 1024 dan DELIMITER adalah " \t\r\n\a". Masuk ke fungsi *main*, terdapat deklarasi *cmdLine* yang merupakan array of char, *command* yang merupakan array of string, *status* bertipe integer, *cwd* yang merupakan array of char sebesar 1024.

```
int main (int argc, char** argv)
{
        char* cmdLine;
        char** command;
        int status;
        char cwd[1024];

        signal(SIGINT, signalhandler);
        signal(SIGTSTP, signalhandler);
```

Pada baris selanjutnya terdapat pemanggilan *signalhandler* untuk memanipulasi pemanggilan SIGINT (ctrl+C), atau SIGSTP (ctrl+D). Pada *signalhandler*, terdapat deklarasi *cwd* lagi seperti di *main*. Selanjutnya terdapat pemanggilan *getcwd(cwd, sizeof(cwd))* yang bertujuan untuk mendapatkan direktori aktif dan menyimpannya ke variabel *cwd*.

```
void signalhandler(int signum) {
        /*do nothing*/
        char cwd[1024];
//      printf("\nERROR: Cannot be stopped by ctrl+c\n");
        getcwd(cwd, sizeof(cwd));
        printf("\ne14shell:%s>> ", cwd);
        fflush(stdout);
}
```

Setelah mendapatkan direktori aktif, program akan mencetak *e14shell:* dilanjut dengan nama direktori yang aktif tersebut. Lalu kembali ke main.

```
int main (int argc, char** argv)
{
        char* cmdLine;
        char** command;
        int status;
        char cwd[1024];

        signal(SIGINT, signalhandler);
        signal(SIGTSTP, signalhandler);

        while(1) {
                getcwd(cwd, sizeof(cwd));

                printf("e14shell:%s>> ", cwd);
                cmdLine=read_line();

                command=split_line(cmdLine);

                if(command[0]==NULL) continue;

                status=execute(command);

                free(cmdLine);
                free(command);
                status=0;
                memset(cwd, 0, sizeof(cwd));
        }
}
```

Di mana dalam fungsi *main*, terdapat *while* yang akan dijalankan secara terus menerus dan berhenti secara otomatis saat user menekan ctrl+D.

Di dalam *while*, juga terdapat pemanggilan *getcwd(cwd, sizeof(cwd))* dan juga dicetak setelah penulisan *e14shell:*. Setelah itu, program meminta inputan dari *user* berupa command yang harus dijalankan. Penginputan command tersebut dimasukkan ke variabel *cmdLine* menggunakan fungsi *read_line()*.

```
char* read_line() {
        int bufsize = BUFFSIZE;
        int position = 0;

        char* buffer=(char*)malloc(sizeof(char)*bufsize);
        char c;

        while(1) {
                c=getchar();
                if(c==EOF){
                        printf("\n");
                        exit(1);
                }
                else if(c=='\n') {
                        buffer[position]='\0';
                        return buffer;
                }
                else buffer[position] = c;

                position = position + 1;
        }
}
```

Pada fungsi *read_line()*, terdapat deklarasi *bufsize* yang bernilai sama dengan *BUFFSIZE*, dan *position* yang bernilai 0. Selanjutnya terdapat deklarasi *buffer* dengan char* yang besarnya ditentukan oleh malloc, dan *c* yang bertipe char. Fungsi *read_line()*, karakter yang diinput akan terus dibaca satu persatu dan dimasukkan ke dalam variabel *c*.

Jika *c* adalah End of File maka program akan berhenti. Jika *c* adalah “\n” maka variabel *buffer* yang berindeks *position* akan berisi ‘\0’. Jika selain itu, *buffer*[*position*] akan terisi dengan karakter yang ada di dalam *c.*

Setelah membaca isi dari seluruh perintah, command tersebut akan dipisahkan sesuai dengan *DELIMITER* yang sudah didefine sebelumnya menggunakan fungsi *split_line(cmdLine)* yang hasil akhirnya akan masuk ke variabel *command.*

```
char** split_line(char* line) {
        int bufsize = BUFFSIZE;
        int position = 0;
        char** tokens = malloc(sizeof(char)*bufsize);
        char* token;

        token = strtok(line, DELIMITER);
        while(token!=NULL){
                tokens[position]=token;
                position=position+1;

                token=strtok(NULL, DELIMITER);
        }

        tokens[position]=NULL;
        return tokens;
}
```

Pada fungsi *split_line*, *cmdLine* yang didapatkan akan dipisahkan menjadi beberapa array dipisahkan sesuai *DELIMITER*. Misalnya, user menginputkan perintah "cd home", maka *command[0]* berisi cd, dan *command[1]* berisi home.

Setelah kembali ke *main*, *command[0]* akan dicek. Jika *command[0]* adalah NULL, maka akan *continue* tanpa melakukan apapun. Setelah itu *command* akan dijalankan pada fungsi *execute* dengan parameter *command* itu sendiri.

```
int execute(char** args){
        if(strcmp(args[0], "cd")==0){
                if(args[1] == NULL) {
                        fprintf(stderr, "ERROR: expected argument for \"cd\"\n"$
                }
                else {
                        if(chdir(args[1])) perror("ERROR");
                }
        }

        else {
                pid_t pid, wpid;
                int status;

                pid = fork();
                if(pid == 0) {
                        execvp(args[0], args);
                }
                else if (pid<0){
                        printf("error\n");
                }
                else {
                        do {
                        wpid = waitpid(pid, &status, WUNTRACED);
                        } while (!WIFEXITED(status) && !WIFSIGNALED(status));
                }
        }

        return 1;
}
```

Pada fungsi *execute*, *command* akan menjadi variabel *args*. Sebelum benar-benar dieksekusi, *args[0]* akan dicek, jika sama dengan "cd" maka program akan kembali mengecek. Jika *args[1]* sama dengan NULL, program akan mencetak "ERROR: expected argument for cd".

Jika *args[1]* bukan NULL, maka *chdir(args[1])* akan dijalankan. Chdir sendiri berfungsi untuk berpindah direktori aktif ke direktori tujuan dalam hal ini adalah *args[1]*. Jika terjadi kesalahan maka program akan mencetak pesan "ERROR" dengan fungsi perror.

Jika *args[0]* bukan berisi cd, maka program akan menjalankan perintah-perintah di dalam *else*. Terdapat deklarasi *pid* dan *wpid* menggunakan *pid_t* dan *status* bertipe integer. Lalu terdapat pembuatan proses anak menggunakan *pid = fork().*

Jika *pid=0,* yang berarti ada di proses anak, maka perintah akan dijalankan dengan menggunakan *execvp(args[0], args)*. Jika *pid<0*, maka program akan mencetak "error", dan jika *pid>0,* yang berarti sedang berada di proses induk, maka program akan terus menunggu hingga proses anak selesai.

Setelah kembali ke *main*, *cmdLine, command* akan difree, *status* akan kembali bernilai 0, *cwd* akan dikosongkan.

Tahap terakhir adalah memberikan kondisi jika command yang diinputkan diahhiri tanda "&", maka command akan dijalankan secara background, dengan kata lain proses induk tidak akan menunggu proses anak.

Yang pertama adalah dengan mendeklarasikan variable *isBackground* sebagai penanda apakah command yang diinputkan hendak dijalankan secara background atau tidak. *isBackground* bernilai awal 0, menandakan bahwa command tidak dieksekusi secara background.

Berikutnya, kita melakukan pass-by-pointer variable *isBackground* ke fungsi *split_line* , agar jika diakhir string dideteksi karakter "&", maka kita akan merubah nilai variable *isBackground* menjadi 1, menandakan bahwa command hendak dieksekusi secara background.

Kemudian kita dapat menggunakan variable *isBackground* di fungsi execute untuk mengeksekusi command secara background.

Mencari Jumlah Bilangan Prima Kurang dari N

Bilangan prima adalah bilangan yang hanya memiliki dua faktor, yaitu angka 1 dan angka itu sendiri. Program ini akan menerima inputan N yang merupakan suatu integer, lalu program akan mencari bilangan-bilangan prima yang kurang dari N dan menghitung ada berapa jumlahnya. Jumlah tersebut akan di cetak di terminal.

Sebuah program untuk menampilkan jumlah bilangan prima yang kurang dari N dibuat.

```
#include<stdio.h>

int main()
{
        int angka, i, j, counter,count_p=0;
        scanf("%d", &angka);

        for(i=2; i<angka; i++){
                counter=0;
                for(j=1; j<=i; j++){
                        if(i%j==0) counter++;
                }
                if(counter==2) count_p++;
        }

        printf("Jumlah bilangan prima: %d\n", count_p);

        return 0;
}
```

Karena program tersebut belum menggunakan prinsip multithread, kami modifikasi sehingga menggunakan multithread.

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>

struct arg_struct {
        int bil;
        int* count_ptr;
};

void *count_prime(void *args){
        int j, counter=0;
        struct arg_struct* argumen=(struct arg_struct*)args;

        for(j=1; j<=(argumen->bil); j++){
                if((argumen->bil)%j==0) counter++;
        }
        if(counter==2) *(argumen->count_ptr)=*(argumen->count_ptr)+1;
}
```

11)

```
        for(i=2; i<angka; i++){
                args[i].bil=i;
                args[i].count_ptr=&count;
                pthread_create(&thread[i], NULL, count_prime, &args[i]);
        }

        for(i=2; i<angka; i++){
                pthread_join(thread[i], NULL);
        }

        printf("Jumlah bilangan prima: %d\n", count);

        return 0;
}
```

```
int main()
{
        int angka, i, count=0;

        scanf("%d", &angka);

        struct arg_struct* args=(struct arg_struct*)malloc(sizeof(struct arg_st$
        pthread_t* thread=(pthread_t*)malloc(sizeof(pthread_t)*(angka+1));

        for(i=2; i<angka; i++){
                args[i].bil=i;
                args[i].count_ptr=&count;
                pthread_create(&thread[i], NULL, count_prime, &args[i]);
        }

        for(i=2; i<angka; i++){
                pthread_join(thread[i], NULL);
```

Terdapat struct bernama *arg_struct* yang berisi
int bil : bilangan yang akan dicek prima atau bukan
int count_ptr* : pointer of integer yang akan menunjuk ke alamat *count* yang ada di fungsi main, di mana isinya adalah jumlah bilangan prima yang sudah ditemukan.

Pada fungsi main, terdapat deklarasi *angka*, *i*, dan *count=0* yang bertipe data integer. *angka* adalah N yang merupakan inputan dari user, *i* adalah variabel yang akan digunakan pada for, dan *count* merupakan variabel yang menyimpan jumlah bilangan prima yang ada.

Setelah menerima inputan dari user, program akan membuat variabel *args* yang bertipe data *struct arg_struct** yang besarnya tergantung dari inputan user, dan diatur menggunakan *malloc*. Setelah itu, terdapat deklarasi *thread* yang besarnya juga tergantung dari inputan user menggunakan pthread_t.

Pada *looping* menggunakan *for* yang pertama, *for* angka berjalan dari 2 sampai *angka*. *for* ini berfungsi untuk melakukan pengecekan bilangan prima menggunakan thread. Pada argumen baris pertama, *args[i].bil=i* berarti *bil* pada struct *args* yang ke *i* akan bernilai *i*. Setelah itu, *args[i].count_ptr=&count* akan berisi jumlah bilangan prima yang sudah ditemukan saat ini.

Lalu, program akan menjalankan *pthread_create (&thread[i], NULL, count_prime , &args[i])*. Baris tersebut akan membuat *angka* melakukan pengecekan dengan menjalankan thread yang sesuai dengan indeksnya tanpa harus menunggu *angka* sebelumnya selesai dicek.

Pada *count_prime* terdapat variabel *j* yang akan digunakan di dalam for, dan variabel *counter* yang akan berguna saat pengecekan bilangan prima itu sendiri.

Pengecekan akan dilakukan di *count_prime*. Karena parameter **args* pada *count_prime* bertipe void, maka parameter yang di*passing* harus di*typecast* terlebih dahulu.

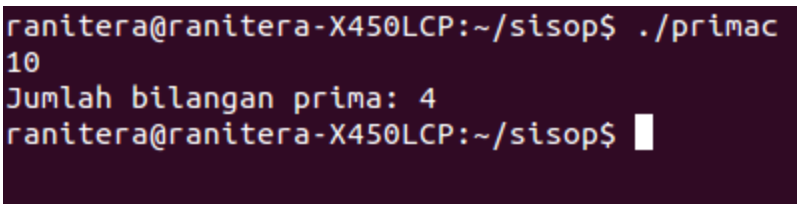
struct arg_struct argumen=(struct arg_struct*) args;*

Setelah itu, masuk ke dalam *looping* dari *j=1 hingga j<=(argumen->bil).* Looping ini berfungsi untuk mengecek apakah angka yang saat ini dicek habis dibagi j (dari 1 sampai angka itu sendiri). Jika angka tersebut habis dibagi, maka *counter* akan bertambah.

Setelah *looping* selesai, *counter* akan dicek. Jika *counter* sama dengan 2 maka angka yang dicek merupakan bilangan prima dan *count* yang ditunjuk oleh *count_ptr* nilainya akan bertambah, jika tidak makan angka tersebut bukan bilangan prima dan tidak terdapat perubahan apapun.

Setelah semua *thread* dijalankan, program akan masuk ke *for* kedua di fungsi *main*. *for* ini berisi *pthread_join(thread[i], NULL)* yang akan dikerjakan dari *i=2* sampai *i=angka* (bilangan-bilangan yang dicek).

Pada akhir program, program akan mencetak jumlah bilangan prima yang kurang dari N.



Membuat Aplikasi Multithread untuk Menyalin Isi File

```
int main () {
        pthread_t t1, t2;
        int status_file1=0;

        pthread_create(&t1, NULL, salin1, &status_file1);
        pthread_create(&t2, NULL, salin2, &status_file1);

        pthread_join(t1, NULL);
        pthread_join(t2, NULL);


        return 0;
}
```

Pada fungsi *main* terdapat deklarasi *t1* dan *t2* menggunakan *pthread_t*, dan *status_file1=0* yang bertipe data integer. Pada baris selanjutnya terdapat pemanggilan thread pertama di *salin1*, dan *salin2*. args ditypecast agar kembali menjadi variabel bertipe integer. *args* ini berisi nilai dari *status_file1*.

```
#include<stdio.h>
#include<pthread.h>

void *salin1(void *args) {
        int* status_file1=(int*)args;

        FILE* inp, *out;
        inp = fopen("tmp.txt", "r");
        out = fopen("salin1_tmp.txt", "w");
        *status_file1=1;

        char kar;

        while(1){
                kar = fgetc(inp);
                if(kar == EOF) break;
                else fputc(kar, out);
        }

        fclose(inp);
        fclose(out);

        *status_file1=2;
}
```

Pada *salin1*, terdapat dua file yang akan digunakan. Pertama, *inp* yang membuka file tmp.txt dan aksesnya adalah “r”. Yang kedua, *out* yang membuka file salin1_tmp.txt dan aksesnya adalah “w”. Saat kedua file tersebut sudah terbuka, makan *status_file* akan bernilai 1.

Setelah itu *kar* yang bertipe *char* akan dideklarasi. Di dalam while, *kar* akan berisi karakter-karakter yang terdapat pada *inp*. Jika menemui End Of File, while akan dihentikan, jika tidak maka karakter-karakter yang terdapat pada *inp* akan disalin ke *out*.

File yang dibuka pada *inp* dan *out*, ditutup. Jika file sudah ditutup, nilai *status_file1* akan berubah menjadi 2.

```
void *salin2(void *args) {
        int* status_file1=(int*)args;

        FILE* inp, *out;

        while(1){
                if(*status_file1==0) continue;
                else {
                        inp = fopen("salin1_tmp.txt", "r");
                        break;
                }
        }
```

```
        out = fopen("salin2_tmp.txt", "w");

        char kar;

        while(1){
                kar = fgetc(inp);
                if(kar == EOF) {
                        if((*status_file1)==1) continue;
                        else break;
                }
                else fputc(kar, out);
        }

        fclose(inp);
        fclose(out);
}
```

Pada *salin2*, program akan melakukan hal yang sama kecuali file yang dibuka dan dalam while. Sebelum membuka file, program akan melakukan pengecekan berulang kali. Jika *status_file1* bernilai 0, yang berarti file pada *salin1* belum dibuka, program akan terus *continue*. Jika sudah tidak bernilai 0, program akan membuka file salin1_tmp.txt berakses “r” yang ada di dalam variabel *inp*, lalu pengecekan akan berhenti.

Setelah salin1_tmp.txt terbuka, *out* adalah variabel yang akan berisi salin2_tmp.txt dengan akses "w" yang terbuka.

Sama seperti pada *salin1*, while berikutnya akan menyalin isi dari *inp* ke *out*. Namun, saat menemui End of File, program akan mengecek terlebih dahulu. Jika *status_file1* bernilai 1 (yang berarti file di *salin1* masih terbuka, program akan lanjut (continue), dan jika *status_file1* tidak bernilai 1 (maksudnya bernilai 2 yang berarti file pada *salin1* sudah ditutup), while akan berhenti.

Setelah selesai, *inp* dan *out* pada *salin2* akan ditutup.

Setelah kembali ke fungsi main, program akan menjalankan *pthread_join(t1, NULL)* dan *pthread_join(t2, NULL)* yang terdapat pada fungsi *main*.

Penggabungan Program

Pada akhrinya, source code untuk mencari jumlah bilangan prima dan menyalin file digabung menjadi satu. Saat menjalankan program gabungan tersebut, user dapat memilih menu untuk keluar dari program, mencari jumlah bilangan prima, atau menyalin file.

```
int main()
{
        int perintah;

        while(1){
                printf("Masukkan Perintah:\n0 Keluar\n");
                printf("1 Mencari Jumlah Bilangan Prima Kurang dari N\n");
                printf("2 Menyalin Isi File\n");
                scanf("%d", &perintah);
                if(perintah==0){
                        printf("Terima Kasih! :D\n");
                        break;
                }
                else if(perintah==1){
                        pthread_t thread1;
                        int angka, i, count=0;

                        printf("Masukkan N: ");
```

```
                if(perintah==0){
                        printf("Terima Kasih! :D\n");
                        break;
                }
                else if(perintah==1){
                        pthread_t thread1;
                        int angka, i, count=0;

                        printf("Masukkan N: ");
                        scanf("%d", &angka);

                        struct arg_struct* args=(struct arg_struct*)malloc(sizeof(struct arg$
                        pthread_t* thread=(pthread_t*)malloc(sizeof(pthread_t)*(angka+1));

                        for(i=2; i<angka; i++){
                                args[i].bil=i;
                                args[i].count_ptr=&count;
                                pthread_create(&thread[i], NULL, count_prime, &args[i]);
                        }
```

```
                struct arg_struct* args=(struct arg_struct*)malloc(sizeof(struct arg$
                pthread_t* thread=(pthread_t*)malloc(sizeof(pthread_t)*(angka+1));

                for(i=2; i<angka; i++){
                        args[i].bil=i;
                        args[i].count_ptr=&count;
                        pthread_create(&thread[i], NULL, count_prime, &args[i]);
                }

                for(i=2; i<angka; i++){
                        pthread_join(thread[i], NULL);
                }

                printf("Jumlah bilangan prima: %d\n\n", count);
        }

        else if(perintah==2){
                pthread_t t1, t2;
```

```
                for(i=2; i<angka; i++){
                        pthread_join(thread[i], NULL);
                }

                printf("Jumlah bilangan prima: %d\n\n", count);
        }

        else if(perintah==2){
                pthread_t t1, t2;
                int status_file1=0;

                pthread_create(&t1, NULL, salin1, &status_file1);
                pthread_create(&t2, NULL, salin2, &status_file1);

                pthread_join(t1, NULL);
                pthread_join(t2, NULL);

                printf("Selesai!\n\n");
        }
```

```
                pthread_create(&t1, NULL, salin1, &status_file1);
                pthread_create(&t2, NULL, salin2, &status_file1);

                pthread_join(t1, NULL);
                pthread_join(t2, NULL);

                printf("Selesai!\n\n");
        }

        else printf("Perintah Tidak Ditemukan!\n\n");
}

return 0;
}
```