

Trabalho de Conclusão de Disciplina

Comparar o desempenho de diferentes kernels quanto a multiprogramação, multiprocessamento e IPC

Equipe:

Gabriel Ribeiro Bernardi - 11821BCC036

Guilherme Soares Correa - 11821BCC026

Introdução

Este Trabalho de Conclusão de Disciplina tem o intuito de comparar o desempenho de diferentes *kernel*s em relação a multiprogramação, multiprocessamento e Inter-Process Communication. Foi utilizado um algoritmo que contém várias estruturas estudadas durante o semestre, tais como, multithread, *signal*, criação de processos, sincronismo entre processos e troca de informações entre os mesmos.

Foram realizados testes em três sistemas operacionais (SOs), comparando entre eles o tempo de processamento, são eles Ubuntu, Solaris e FreeBSD. Dentre os três sistemas testados apenas o Solaris não conseguiu executar o algoritmo, uma vez que o mesmo travava durante a execução.

No algoritmo utilizado não conseguimos concluir as letras B e C do trabalho 3, focamos em apresentar os números e o tempo de execução.

Código

Estruturas de sincronização

Foram utilizados dois processos de sincronização. O primeiro foi o semáforo, implementado nos processos p1, p2, p3 e p4 para gerenciar o acesso à fila 1.

Exemplo em código:

```
sem_wait((sem_t*)&fila_shared->mutex);  
sem_post((sem_t*)&fila_shared->mutex);
```

Outra estrutura utilizada foi o *busy wait*, que faz o gerenciamento dos processos p5, p6 e p7 em relação ao acesso à fila 2.

Exemplo em código:

```
while ( fila_shared2->signal != 0 ){if(fila_shared2->StopAllProcess==0)exit(0);}
```

Neste exemplo o processo ficará em *loop* até chegar sua vez. O tratamento interno da estrutura de repetição, foi feito para que caso o programa termine o processo finalize junto, evitando processos pendentes no sistema.

Signal

O sinal foi utilizado para que quando os processos p1, p2 ou p3 terminarem de preencher a fila, o p4 comece o seu trabalho de retirada de valores da fila. Em código, sua implementação foi a seguinte:

```
signal(SIGUSR1, sinal); |  
kill(getpid(),SIGUSR1);
```

Função que será chamada chamada:

```
void sinal(int p)
{
    fila_shared->sinal = 1;
}
```

Criação dos processos.

Foi feito um 'laço' de tamanho 7. Dentro dele foi feito um 'fork' para cada processo.

Exemplo:

```
for(int i=0;i<7;i++) // loop will run n times (n=5)
{
    if (i==0){
        if(fork() == 0)
        {
            //p1
            int cont=0;
            addfila(NULL);
            exit(0);
        }
    }
}
```

Para os threads foi utilizado da seguinte forma:

```
// p4
pthread_create(&thread4, NULL, transferePorPipe2, NULL); // inicia e executa o thread criado
transferePorPipe(NULL);
pthread_join(thread4, NULL); // finaliza
exit(0);
```

No caso de p4 que possui duas threads, primeiro é declarado uma nova thread com o 'pthread_create' e logo após é chamado a função "transferePorPipe" na thread que é criada durante a utilização da estrutura 'fork'.

Detalhamento de cada um dos processos.

P1, P2 e P3.

Os três processos competem entre si para escrever na fila 1 (em memória compartilhada).

Função utilizada pelos três processos:

```

void* addfila ( void *ptr){
    int num=0;
    while(fila_shared2->StopAllProcess==1){

        sem_wait((sem_t*)&fila_shared->mutex);
        if (fila_shared->sinal == 0){
            if (fila_shared->nItens<9){// wait clear the fifo
                fila_shared->nItens++;
                num=rand()%1000;
                fila_shared->dados[fila_shared->nItens] = (num==0? 1:num);
            }
            if (fila_shared->nItens>8) {
                kill(getpid(),SIGUSR1);
            } // signal to p4
        }
        sem_post((sem_t*)&fila_shared->mutex);
    }
    pthread_exit(0); /* exit thread */
}

```

P4.

Esse processo tem 2 threads, cada uma escrevendo em pipes diferentes.

```

void* transferePorPipe( void *ptr){
    printf("pid 1: %d\n",getpid());
    while (fila_shared2->StopAllProcess==1){
        sem_wait((sem_t*)&fila_shared->mutex); // SEMA
        if (fila_shared->sinal == 1 && fila_shared->nItens >-1){
            write(canal1[1],&fila_shared->dados[fila_shared->nItens],sizeof(int)); // writing on chanel, conection of p4 and p5
            fila_shared->nItens--;
            if(fila_shared->nItens==1){
                clearfifo(fila_shared );
            }
        }
        sem_post((sem_t*)&fila_shared->mutex);
    }
    close(canal1[1]);
    pthread_exit(0); /* exit thread */
}

```

Thread 1

```

void* transferePorPipe2( void *ptr){
    while (fila_shared2->StopAllProcess==1){
        sem_wait((sem_t*)&fila_shared->mutex); // SEMA
        if (fila_shared->sinal == 1 && fila_shared->nItens >-1){
            write(canal2[1],&fila_shared->dados[fila_shared->nItens],sizeof(int)); // writing on chanel, conection of p4 and p6
            fila_shared->nItens--;
            if(fila_shared->nItens==1){
                clearfifo(fila_shared);
            }
        }
        sem_post((sem_t*)&fila_shared->mutex);
    }
    close(canal2[1]);
    pthread_exit(0); /* exit thread*/
}

```

Thread 2:

As duas funções seguem o mesmo padrão. Primeiro é utilizado a estrutura de semáforo, representado por 'sem_wait'. Logo após é verificado se p1, p2 ou p3 mandaram o sinal de que a fila está cheia e que ele pode retirar valores e escrever

no seu respectivo canal. No final do processo, a fila é limpa e é avisado os processos p1, p2 e p3 que podem voltar a produzir números aleatórios.

P5 e P6.

Cada um desses processos é mono thread, porém funcionam de forma semelhante, segue:

```
void* readp5( void *ptr ){
    int val, n=0;
    while(fila_shared2->StopAllProcess==1){
        while ( fila_shared2->sinal != 0 ){if(fila_shared2->StopAllProcess==0)exit(0);}
        if (fila_shared2->nItens<9) {
            fila_shared2->nItens++;
            n=read(canal1[0],&fila_shared2->dados[fila_shared2->nItens],sizeof(int));
            if (n==-1){
                printf("Erro!!");
                exit(0);
            }
        }
        changesinalf2(fila_shared2,1);
    }
    close(canal1[0]);
    pthread_exit(0); /* exit thread */
}
```

Utilizada em P5:

```
void* readp6( void *ptr ){
    int val,n=0;
    while(fila_shared2->StopAllProcess==1){
        while ( fila_shared2->sinal != 1 ){if(fila_shared2->StopAllProcess==0)exit(0);}
        if (fila_shared2->nItens<9) {
            fila_shared2->nItens++;
            n=read(canal2[0],&fila_shared2->dados[fila_shared2->nItens],sizeof(int));
            if (n==-1){
                printf("Erro!!");
                exit(0);
            }
        }
        changesinalf2(fila_shared2,2);
    }
    close(canal2[0]);
    pthread_exit(0); /* exit thread */
}
```

Utilizada em P6:

Ambos funcionam da seguinte forma, esperam a sua vez de funcionar através da estrutura Busy wait, representado pelo 'while (fila_shared2->sinal != X)'. Além disso é feito o controle para que seja escrito na fila apenas o limite de 10 valores.

P7.

P7 funciona com 3 threads, construídos da seguinte forma:

```
// p7
pthread_create(&thread7[0], NULL, result3, NULL); // inicia e executa o thread criado
pthread_create(&thread7[1], NULL, result2, NULL); // inicia e executa o thread criado
result(NULL);
pthread_join(thread7[0], NULL);
pthread_join(thread7[1], NULL);
exit(0);
```

Rotina de cada uma de suas threads:

```
void* result( void *ptr ){
    int cont=-1;
    while (fila_shared2->StopAllProcess==1){
        while ( fila_shared2->sinal != 2 ){if(fila_shared2->StopAllProcess==0)exit(0);} // Busy wait
        if(fila_shared2->StopAllProcess==0)exit(0);
        if(fila_shared2->nItens>-1){
            printf("1 - numero: %d, %d\n",fila_shared2->dados[fila_shared2->nItens],fila_shared2->totnum);
            fflush(stdout);
            fila_shared2->nItens--;
            fila_shared2->totnum++;
            if (fila_shared2->totnum==10000){
                relp7();
                fila_shared2->StopAllProcess=0; // response to stop all process build
            }
        }
        changesinalf2(fila_shared2,3);
    }
    pthread_exit(0); /* exit thread */
}
```

Thread 1

```
void* result2( void *ptr ){
    int cont=-1;
    while (fila_shared2->StopAllProcess==1){
        while ( fila_shared2->sinal != 3 ){if(fila_shared2->StopAllProcess==0)exit(0);} // Busy wait
        if(fila_shared2->StopAllProcess==0)exit(0);
        if(fila_shared2->nItens>-1){
            printf("2 - numero: %d, %d\n",fila_shared2->dados[fila_shared2->nItens],fila_shared2->totnum);
            fflush(stdout);
            fila_shared2->nItens--;
            fila_shared2->totnum++;
            if (fila_shared2->totnum==10000){
                relp7();
                fila_shared2->StopAllProcess=0; // response to stop all process build
            }
        }
        changesinalf2(fila_shared2,4);
    }
    pthread_exit(0); /* exit thread */
}
```

Thread 2

```

void* result3( void *ptr ){
    int cont=-1;
    while (fila_shared2->StopAllProcess==1){
        while ( fila_shared2->senal != 4 ){if(fila_shared2->StopAllProcess==0)exit(0);} // Busy wait
        if(fila_shared2->StopAllProcess==0)exit(0);
        if(fila_shared2->nItens>-1){
            printf("3 - numero: %d, %d\n",fila_shared2->dados[fila_shared2->nItens],fila_shared2->totnum);
            fflush(stdout);
            fila_shared2->nItens--;
            fila_shared2->totnum++;
            if (fila_shared2->totnum==10000){
                relp7();
                fila_shared2->StopAllProcess=0;           // response to stop all process build
            }
        }
        changesinalf2(fila_shared2,0);
    }
    pthread_exit(0); /* exit thread */
}

```

Thread 3

Execução e desempenho

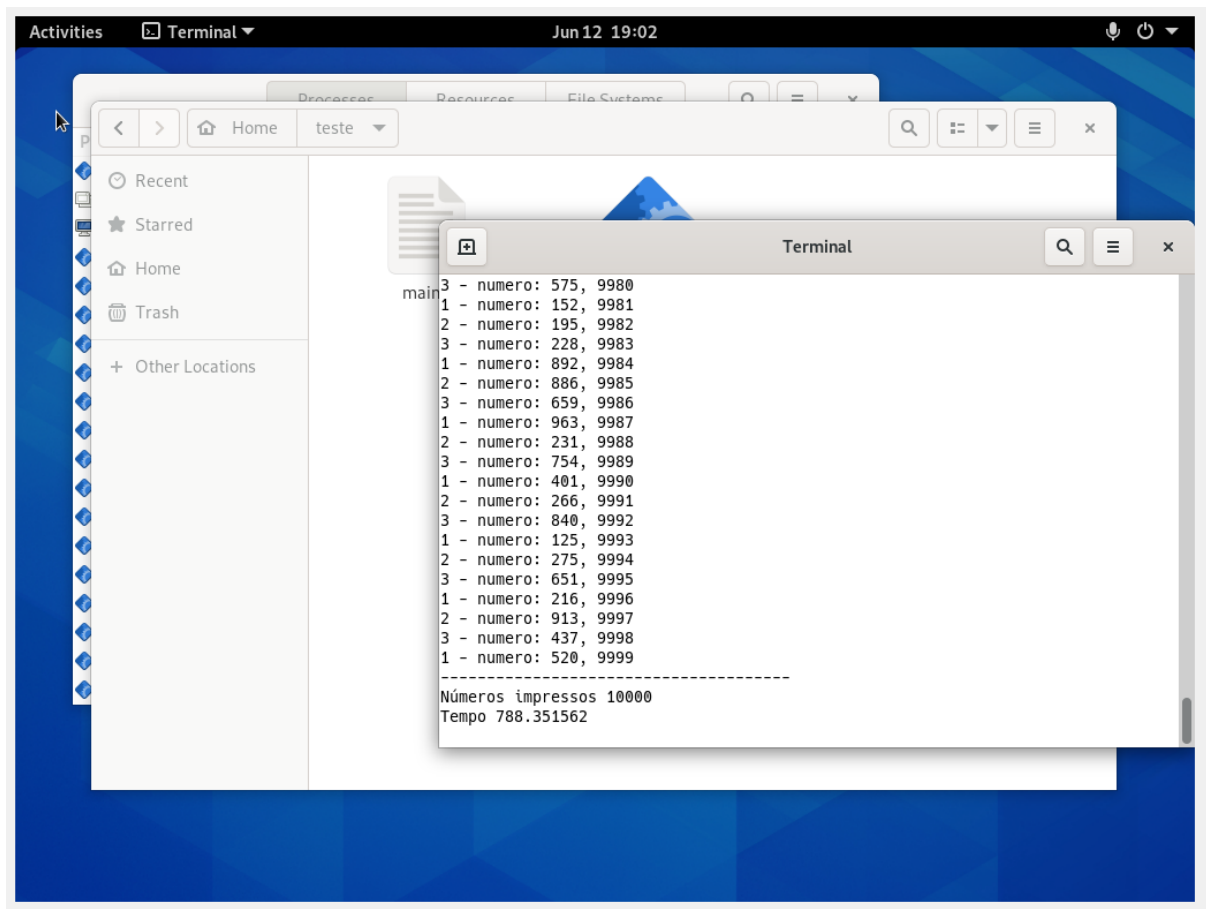
Para a execução do código foram utilizados três sistemas operacionais *UNIX Like*. São eles: Linux Ubuntu 20.04.2.0, Solaris 11.4 e FreeBSD 13.0. Ambos foram instalados em ambiente de máquinas virtuais utilizando VirtualBox 6.1 via Windows 10 21H1. A configuração padrão do computador utilizado foi: Processador AMD Ryzen 5 3400G 3.7GHz com turbo speed de 4.2GHz, com 4 núcleos físicos e 8 threads, 16GB de memória RAM. No entanto, como o algoritmo foi executado em máquinas virtuais, a configuração destinada para as mesmas foi: 4 CPUs, 3072 MB de memória RAM. O código foi executado 11 vezes em cada um dos SOs, no entanto o tempo gasto na primeira execução foi descartado para maior confiabilidade dos resultados. Os resultados obtidos podem ser observados imagem a seguir:

	A	B	C	D
1		FreeBSD	Ubuntu	Solaris
2	1	798.031250	125.962053	
3	2	772.828125	122.137326	
4	3	788.351562	122.463212	
5	4	787.754543	121.962771	
6	5	786.265788	126.751875	
7	6	797.258683	125.212133	
8	7	784.092310	120.547576	
9	8	787.989142	125.968953	
10	9	777.090488	117.476416	
11	10	792.252190	115.020524	
12	MÉDIA	787.191408	122.350284	undefined

Tabela comparativa com tempos de execução

Com os dados obtidos, pode-se observar que o tempo médio de execução do algoritmo no FreeBSD foi de 787 segundos aproximadamente. Já no Ubuntu, o

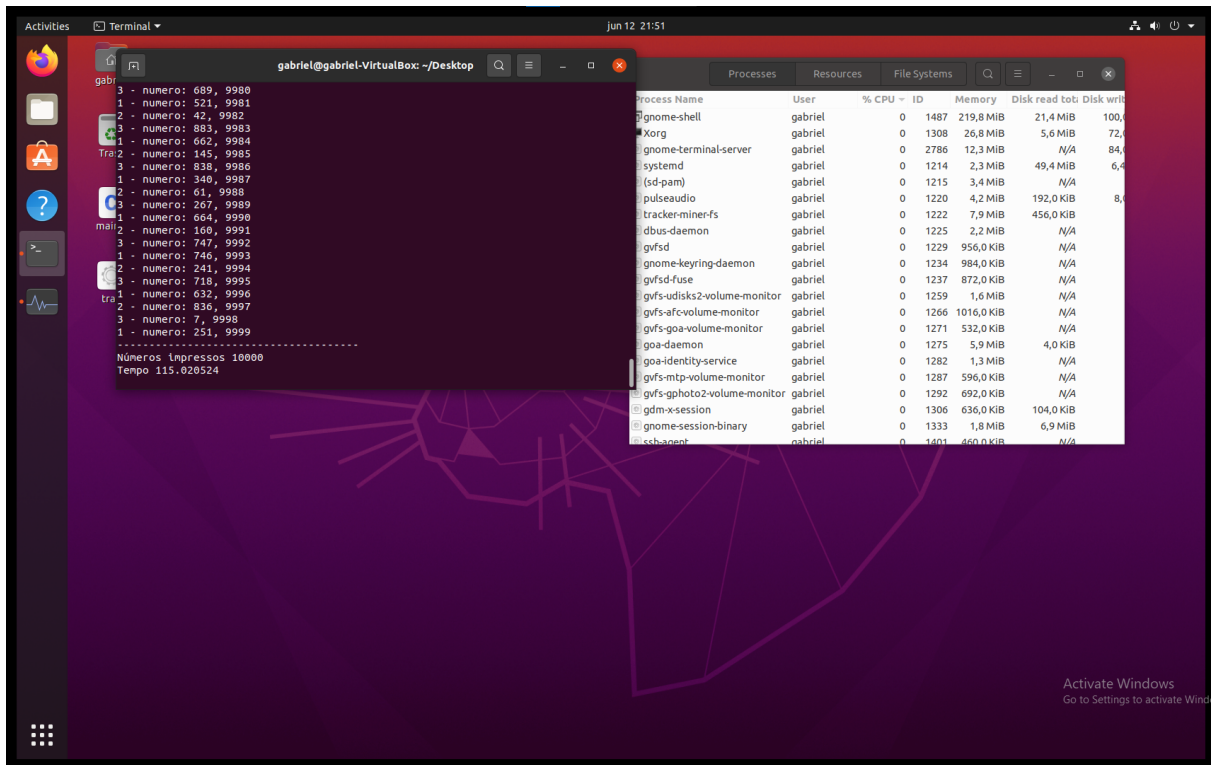
tempo médio de execução foi de 122 segundos, o que torna a execução no FreeBSD cerca de 6,43 vezes mais lenta. Em relação ao Solaris, a instalação do SO foi feita, no entanto o código não estava conseguindo ser executado por completo resultando na não comparação do tempo de execução dele com os demais.



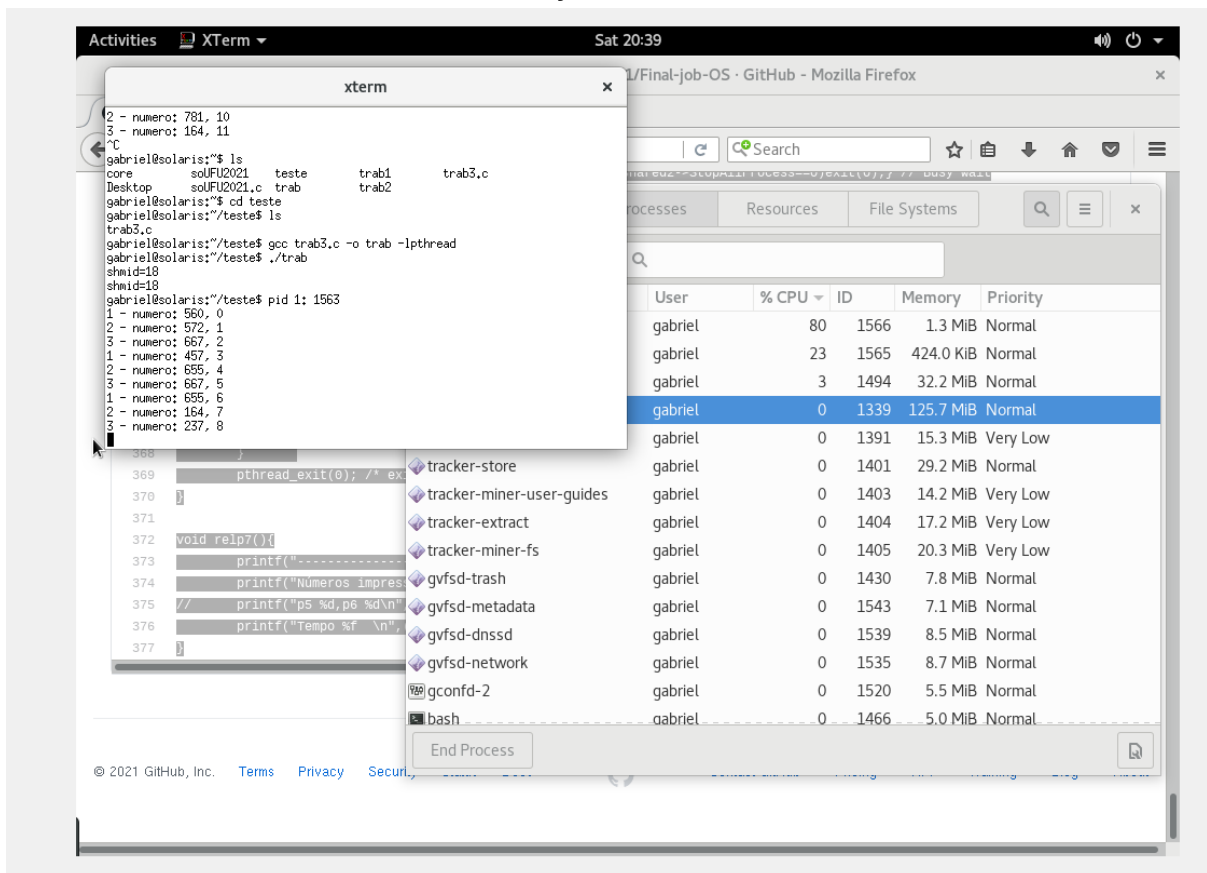
The screenshot shows a FreeBSD desktop environment with a blue background. A terminal window is open, displaying the output of a program. The output consists of 30 lines of text, each showing a number and a time value. The numbers are grouped by a 'main' label and a 'numero' prefix. The times are in seconds, with some values being very small (e.g., 0.000000) and others being larger (e.g., 0.000001). The terminal window is titled 'Terminal' and has a search icon and a menu icon. A file manager window is also visible in the background, showing a directory structure with 'Recent', 'Starred', 'Home', and 'Trash' folders.

```
main
3 - numero: 575, 9980
1 - numero: 152, 9981
2 - numero: 195, 9982
3 - numero: 228, 9983
1 - numero: 892, 9984
2 - numero: 886, 9985
3 - numero: 659, 9986
1 - numero: 963, 9987
2 - numero: 231, 9988
3 - numero: 754, 9989
1 - numero: 401, 9990
2 - numero: 266, 9991
3 - numero: 840, 9992
1 - numero: 125, 9993
2 - numero: 275, 9994
3 - numero: 651, 9995
1 - numero: 216, 9996
2 - numero: 913, 9997
3 - numero: 437, 9998
1 - numero: 520, 9999
-----
Números impressos 10000
Tempo 788.351562
```

Execução no FreeBSD



Execução no Ubuntu



Tentativa de execução no Solaris