Experimento #1
Sistemas Operacionais A

Adriano de Oliveira Munin	RA:17066960
Fábio Seiji Irokawa	RA:17057720
Lucas Rodrigues Coutinho	RA:17776501
Marcos Lelis de F. Oliveira	RA:16248387
Paulo M. Birocchi	RA:16148363

Introdução

O experimento realizado se refere à aprendizagem na criação de processos pai e filho através do comando “fork()” e o conceito de tempo de acesso de um programa executado no computador, o qual não é previsível determinar um tempo específico de seu acesso devido ao grande número de variáveis.

Além do experimento, a introdução ao sistema operacional Linux será necessário para futuros experimentos devido a sua versatilidade e mais liberdade na hora de programar e utilizar suas ferramentas comparada com o Windows, que é um pouco mais limitado.

A introdução ao uso dos comandos “fork()”, “kill()” ,”usleep()” e “exec()” permite a manipulação e controle na hora da chamada dos processos, a hora da “dormência”, finalização e execução dos mesmos durante a execução do programa, o que possibilita a fácil manipulação dos processos, tanto o pai quanto o filho.

O experimento também ajudará o grupo a ter experiência na hora de programar utilizando os processos, além de ser uma introdução à matéria de Sistemas Operacionais.

TAREFAS

PRIMEIRA TAREFA:

Erros presente no programa:

// Biblioteca stdlib.h não estava incluída, a biblioteca foi
#include <stdlib.h> incluída.

// SLEEP_TIME deve ser do tipo float e estava definida como int, foi colocado .0 após o valor.
#define SLEEP_TIME 640.0

// A variável rtn não estava com o tipo definido, foi definida como int.
rtn = 1;

//rtn precisa ser diferente de 0, indicando que é um processo pai, para assim poder criar um filho, a comparação de igualdade foi alterado para desigualdade.

```
for( count = 0; count < NO_OF_CHILDREN; count+- ) {
        if( rtn == 0 ) {
                rtn = fork();
        }
        else {
                break;
        }
}
```

//rtn é uma variável, não deve ser usado como função, foi removido o parêntese.
if(rtn() == 0)

Respostas das questões

Perguntas fora do código

1) gcc experimento1.c -o experimento1

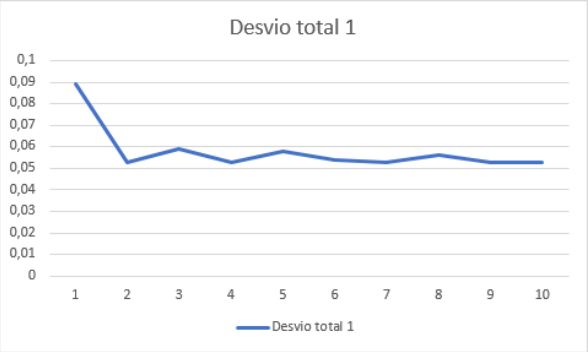
2) A diretiva ‘&’ permite ao usuário que execute um comando definido por ele em segundo plano, tornando possível a execução de diversos processos.

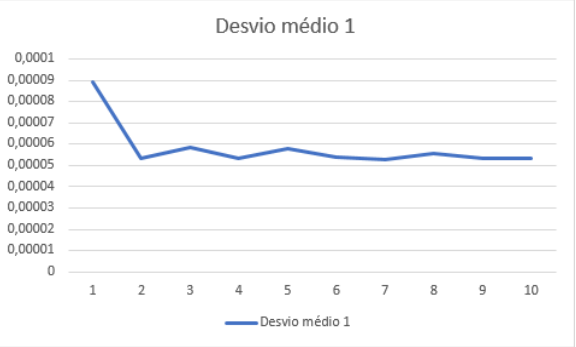
3) O comando ./ indica que o usuário quer executar um programa que está no diretório atual. Para executar o GCC não é necessária a utilização do "./" pois o GCC é um comando utilizado para realizar a compilação do programa em questão.

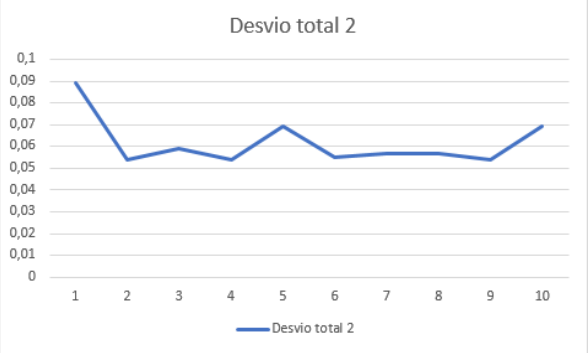
4) A CPU utilizada no experimento foi um Core i7 com 8 MB de cache, 3,8 GHz de 4 núcleos físicos.

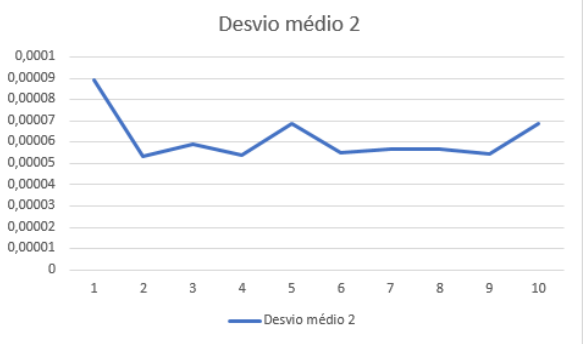
Tabela 01- valores obtidos nos teste de desvio médio e total, com variação de carga

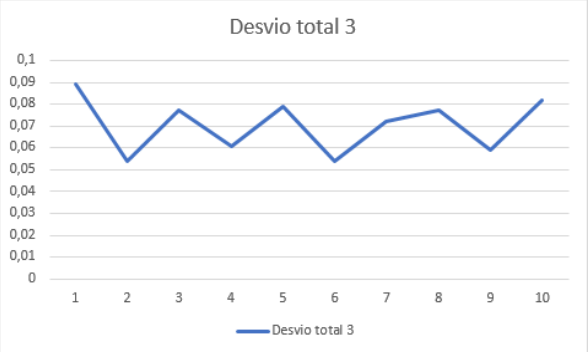
Rodada	Filho #1		Filho #2		Filho #3	
	Desvio total 1	Desvio médio 1	Desvio total 2	Desvio médio 2	Desvio total 3	Desvio médio 3
01 (00 Carga)	0.089	0.00008942	0.089	0.00008930	0.089	0.00008938
02 (05 Cargas)	0.053	0.00005334	0.054	0.00005352	0.054	0.00005381
03 (10 Cargas)	0.059	0.00005861	0.059	0.00005914	0.077	0.00007725
04 (15 Cargas)	0.053	0.00005326	0.054	0.00005382	0.061	0.00006114
05 (20 Cargas)	0.058	0.00005787	0.069	0.00006851	0.079	0.00007851
06 (25 Cargas)	0.054	0.00005386	0.055	0.00005485	0.054	0.00005432
07 (30 Cargas)	0.053	0.00005294	0.057	0.00005700	0.072	0.00007246
08 (35 Cargas)	0.056	0.00005575	0.057	0.00005685	0.077	0.00007741
09 (40 Cargas)	0.053	0.00005344	0.054	0.00005447	0.059	0.00005913
10 (45 Cargas)	0.053	0.00005347	0.069	0.00006891	0.082	0.00008189


Desvio total 1
0,1
0,09
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
1 2 3 4 5 6 7 8 9 10
Desvio total 1


Desvio médio 1
0,0001
0,00009
0,00008
0,00007
0,00006
0,00005
0,00004
0,00003
0,00002
0,00001
0
1 2 3 4 5 6 7 8 9 10
Desvio médio 1


Desvio total 2
0,1
0,09
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
1 2 3 4 5 6 7 8 9 10
Desvio total 2


Desvio médio 2
0,0001
0,00009
0,00008
0,00007
0,00006
0,00005
0,00004
0,00003
0,00002
0,00001
0
1 2 3 4 5 6 7 8 9 10
Desvio médio 2


Desvio total 3
0,1
0,09
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
1 2 3 4 5 6 7 8 9 10
Desvio total 3


Desvio médio 3
0,0001
0,00009
0,00008
0,00007
0,00006
0,00005
0,00004
0,00003
0,00002
0,00001
0
1 2 3 4 5 6 7 8 9 10
Desvio médio 3

Tabela 02 - Valores de tempo obtidos sem cargas com variação de tempo de dormência

Tempo dormência	Tempo Obtido	Tempo Esperado	Desvio Tempo
40	0.190246	0.08000	0.110246
40	0.190195	0.08000	0.110195
80	0.269991	0.16000	0.109991
80	0.270345	0.16000	0.110423
160	0.434136	0.32000	0.114136
160	0.434672	0.32000	0.114672
320	0.766116	0.64000	0.126116
320	0.766013	0.64000	0.126013
640	1.134602	1.28000	0.154602
640	1.435339	1.28000	0.155333

Segunda Tarefa:

Tabela 03 - Valores de desvio total e médio de filhos em processos separados

Rodada	Filho #1		Filho #2		Filho #3		Filho #4		Filho #5	
	Desvio total	Desvio médio	Desvio total	Desvio médio	Desvio total	Desvio médio	Desvio total	Desvio médio	Desvio total	Desvio médio
1	0.154	0.00008	0.168	0.00008	0.169	0.00008	0.175	0.00009	0.173	0.00009
2	0.149	0.00007	0.169	0.00008	0.170	0.00009	0.174	0.00009	0.172	0.00009
3	0.159	0.00008	0.177	0.00009	0.177	0.00009	0.176	0.00009	0.178	0.00009
4	0.154	0.00008	0.178	0.00009	0.178	0.00009	0.178	0.00009	0.179	0.00009
5	0.149	0.00007	0.168	0.00008	0.174	0.00009	0.175	0.00009	0.178	0.00009
6	0.147	0.00007	0.163	0.00008	0.172	0.00009	0.174	0.00009	0.177	0.00009
7	0.156	0.00008	0.178	0.00009	0.177	0.00009	0.179	0.00009	0.181	0.00009
8	0.148	0.00007	0.168	0.00008	0.170	0.00008	0.165	0.00008	0.164	0.00008
9	0.146	0.00007	0.161	0.00008	0.162	0.00008	0.167	0.00008	0.166	0.00008
10	0.143	0.00007	0.162	0.00008	0.162	0.00008	0.168	0.00008	0.168	0.00008

5) Um processo determinístico é nada mais que um processo previsível, isto é, o usuário pode ter uma previsão da saída apresentada a ele no final da execução do processo.

6) Para conseguir informações de processos em determinado instante, basta utilizar ferramentas já disponibilizadas pelo próprio sistema operacional, como o Gerenciador de Tarefas (Windows) e o Monitor do Sistema (Linux), sendo possível consultar tanto o PID do processo, como seu uso de CPU, memória RAM e disco.

Perguntas dentro do código

1) O compilador cria uma cópia do arquivo-interface da biblioteca, para que o programa tenha acesso a ela.

2) A biblioteca stdio.h possui principalmente funções para operações de entrada/saída, como fazer leitura do teclado ou imprimir informações na tela, por exemplo: printf(), scanf(), fprintf(), fscanf().

3) O include tem a função de fazer de informar ao compilador para incluir um arquivo específico quando for realizada a compilação do código principal, ou seja, sua função basicamente é fazer a inclusão de um outro código dentro do código principal.

4) Os parâmetros **argc** e **argv** dão ao programador acesso à linha de comando com a qual o programa foi chamado.

O **argc** (argument count) é um inteiro e possui o número de argumentos com os quais a função **main()** foi chamada na linha de comando. Ele é, no mínimo 1, pois o nome do programa é contado como sendo o primeiro argumento.

O **argv** (argument values) é um ponteiro para uma matriz de strings. Cada string desta matriz é um dos parâmetros da linha de comando. O **argv[0]** sempre aponta para o nome do programa. É para saber quantos elementos temos em **argv** que temos **argc**.

5) Em teoria a relação entre o SLEEP_TIME e o desvio não existe, pois a imprecisão esta medida em unidade de tempo e nao em razao ao tempo total. No entanto não foi o que se verificou, pois obtivemos exatamente o resultado oposto ao previsto, sendo quanto maior o SLEEP_TIME, maior o desvio.

Análise dos Resultados

Com base nos dados obtidos verificamos que não há uma lógica direta entre a carga aplicada ao processador e o desvio obtido no programa exemplo, pois nos sistemas operacionais modernos há diversos processos e tecnologias para melhorias de desempenho, muitos deles focados em escalonamento de processos para o uso da CPU. Com isso, acabamos obtendo resultados que não esperávamos, como a diminuição do desvio com o aumento da carga dos processadores. Para tentar minimizar os efeitos de tecnologias e processos que alteram a fila de execução de processos, compilamos o Kernel Linux com diversos itens desativados, entre eles o **Advanced Programmable Interrupt Controller** (**APIC**), ***ACPI (Advanced Configuration and Power Interface), Multicore Schedule entre*** *outros. Com isso obtivemos resultados mais próximo do esperado, no entanto, embora a diferença dos desvios entre os testes com e sem carga nos processadores tenha diminuído, ainda obtivemos desvios menores com cargas sendo executadas, porém, isso se deve a mais fatores como por exemplo as chamadas de sistemas antes mesmo de acabar o Quantum do processador.*

Conclusão

Com a realização deste experimento foi possível atingir os objetivos definidos, tal como entender e assimilar a prática da realização de futuros experimentos e a confecção dos respectivos relatórios, além disso, aperfeiçoamos a compreensão da utilização do GCC.

Os maiores benefícios que este experimento trouxe foi o de compreender mais a fundo diversos mecanismos e processos de controles que se tem em um sistema operacional Unix, dentre eles, a criação de processos, medição de processos, multitasking, escalonamento de processos, threads entre outros.

Uma curiosidade que ocorreu no experimento foi a influência que o Quantum (tempo em que um processo fica sendo executado no processador) teve nos testes do programa da Tarefa 1 com cargas, no qual obtivemos resultados contra-intuitivos. Na ocorrência de processos que requer bastante uso de processador, o desvio da medição de tempo de nosso programa diminuiu, que era algo não esperado. Com tal resultado, e sabendo que um dos grandes influenciadores disso e o **Advanced Programmable Interrupt Controller** (**APIC**), uma parte do Unix que controla o Quantum do processador, compilamos o kernel do Linux com tal opção desativada, instalamos em uma máquina e refizemos os teste, com isso, obtivemos resultados bem interessantes. Notamos que houve um aumento na variação dos dados, mas mesmo assim, a diferença do desvio entre a execução do programa sem carga e com carga foi menor, no entanto, com carga, o processo com carga ainda continuou com um desvio menor. Com isso observado, concluímos que há mais sistemas do sistema operacional que influenciam tais resultados, assim como chamadas de sistemas dos próprios processos sendo executados que acabam por terminar seu tempo de execução antes do quantum acabar.