

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS – PUC-**  
**CAMPINAS**

***Experimento 1***  
***Sistemas Operacionais A***

<b>ALUNO</b>	<b>RA</b>
Beatriz Morelatto Lorente	18071597
Cesar Marrote Manzano	18051755
Fabricio Silva Cardoso	18023481
Pedro Ignácio Trevisan	18016568

**-= Sumário =-**

<b>1 Introdução.....</b>	<b>3</b>
<b>2 Apresentação dos erros do programa exemplo e suas soluções.....</b>	<b>4</b>
<b>3 Resultados da execução do programa exemplo.....</b>	<b>6</b>
<b>4 Resultados da execução do programa modificado.....</b>	<b>8</b>
<b>5 Respostas das perguntas.....</b>	<b>10</b>
<b>6 Análise dos Resultados.....</b>	<b>12</b>
<b>7 Conclusão.....</b>	<b>17</b>

## **-= Introdução =-**

O experimento realizado permite o entendimento de dois conceitos amplamente usados em Sistemas Operacionais: o de criação de processos e o conceito de tempo. O experimento está dividido em duas tarefas, nas quais pode-se observar a duração de um trecho de programa variando o número de filhos criados, o tempo de dormência e a forma como se relaciona pai e filho entre as duas tarefas.

A primeira tarefa teve como objetivo visualizar o desvio total e o desvio médio de processos filhos de um trecho de programa. Foram realizadas 10 rodadas de teste, sendo que a primeira foi executada apenas o programa principal, sem nenhum programa rodando em background, e a partir da segunda tarefa foi executado o programa principal e 5 programas que consumiam CPU (a cada rodada eram acrescentados mais 5 programas desse tipo).

Na segunda tarefa o programa foi modificado criando dois programas, um que caracteriza o processo pai onde é feito o "fork()" para criação dos filhos, e outro que funciona como filho onde é calculado os desvios (total e médio). Essa tarefa foi executada dez vezes variando o tempo de "SLEEP\_TIME", de forma que a variação do mesmo fosse múltiplo de 200ms, iniciando em 400ms. Na segunda parte da segunda tarefa, foi pedido para que o processo pai enviasse um sinal de kill para cada filho, para assim analisarmos o comportamento do programa.

## **-= Apresentação dos erros do programa exemplo e suas soluções -=**

Ao compilar o programa exemplo foi possível perceber alguns erros de sintaxe e lógica do programa. Os problemas estão listados abaixo, seguidos de suas soluções (as correções estão destacadas em negrito e itálico).

### **Problema 1, 2 e 3**

```
rtn = 1;
for ( count = 0; count < NO_OF_CHILDREN; count+- ) {
    if( rtn == 0 ) {
        rtn = fork();
    } else {
        break;
    }
}
```

Nesse trecho de código percebe-se três erros de código. O primeiro é a não declaração da variável "rtn". O segundo é a incrementação incorreta do valor da variável "count". O terceiro está presente dentro do comando if. Esse problema deve-se ao fato que dentro do comando if há um fork(), que só poderia ser feito se o valor de rtn fosse igual a 1, uma vez que o valor do fork é diferente de zero, representa um filho.

#### **Programa corrigido:**

```
pid_t rtn = 1;
for (count = 0; count < NO_OF_CHILDREN; count++) {
    if (rtn != 0) {
        rtn = fork();
    } else {
        break;
    }
}
```

### **Problema 4**

```
if (rtn() == 0)
```

Como rtn não é uma função e sim uma variável o comando if está incorreto.

#### **Problema corrigido:**

```
if (rtn == 0)
```

Apenas retiramos os parênteses para corrigir o problema.

### **Problema 5**

```
printf("Filho  #%d  --  desvio  total:  %.3f  --  desvio  medio:  %.1f\n",child_no,  drift
NO_OF_ITERATIONS*SLEEP_TIME/MICRO_PER_SECOND,(drift
NO_OF_ITERATIONS*SLEEP_TIME/MICRO_PER_SECOND)/NO_OF_ITERATIONS);
```

O trecho de código em si está correto, porém com a quantidade de casas decimais usadas, não conseguiríamos analisar os desvios.

### Problema corrigido:

```
printf("Filho  #%d  --  desvio  total:  %.8f  --  desvio  medio:  %.8f\n",child_no,  drift  
NO_OF_ITERATIONS*SLEEP_TIME/MICRO_PER_SECOND,(drift  
NO_OF_ITERATIONS*SLEEP_TIME/MICRO_PER_SECOND)/NO_OF_ITERATIONS);
```

Para podermos analisar os resultados, colocamos a precisão de 8 casas decimais nos desvios.

### Problema 6

```
for (count = 0; count > NO_OF_CHILDREN; count ++ ) {  
    wait(NULL);  
}
```

Como a condição do for é “count > NO\_OF\_CHILDREN”, isso causaria filhos zumbis, acontecimento que não é esperado ocorrer.

### Problema corrigido:

```
for (count = 0; count < NO_OF_CHILDREN; count ++ ) { wait(NULL); }
```

Para isso modificamos o sinal de maior (>) para menor (<).

## **-= Resultados da execução do programa exemplo -=**

### ***Execuções do programa exemplo com CPU carregada gradualmente***

Rodada	Filho #1		Filho #2		Filho #3	
	Desvio total	Desvio médio	Desvio total	Desvio médio	Desvio total	Desvio médio
1	0.14078701	0.00014079	0.13958097	0.00013958	0.14059603	0.00014060
2	0.06702602	0.00006703	0.06812000	0.00006812	0.06951094	0.00006951
3	0.05772305	0.00005772	0.05259705	0.00005260	0.05322897	0.00005323
4	0.05671597	0.00005672	0.05260205	0.00005260	0.05437005	0.00005437
5	0.05682099	0.00005682	0.05342400	0.00005342	0.05260003	0.00005260
6	0.05608106	0.00005608	0.05280805	0.00005281	0.05259800	0.00005260
7	0.05663896	0.00005664	0.05280805	0.00005281	0.05282104	0.00005282
8	0.05694401	0.00005694	0.05280304	0.00005280	0.05280697	0.00005281
9	0.05558395	0.00005558	0.05280900	0.00005281	0.05281496	0.00005281
10	0.05420494	0.00005420	0.05270505	0.00005271	0.05281198	0.00005281

Fonte do programa usado para roubar tempo da CPU (carga.c):

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

```
int main ()
{
    int i = 0;
    while (1)
    {
        i++;
    }

    return 0;
}
```

O programa foi usado a partir da segunda rodada de teste, já iniciando com 5 cargas sendo executados ao mesmo tempo. A cada rodada foi sendo acrescentado mais 5 cargas para verificarmos o efeito no desvio.

Também foi feito um teste alterando duas constantes do programa: NO\_OF\_ITERATIONS e SLEEP\_TIME, sendo que a primeira foi alterada para 5000 e a segunda para 2000. Os testes foram realizados sem carga na CPU. As tabelas abaixo mostram o resultado das alterações.

### ***Execuções do programa exemplo com variáveis alteradas***

Filho	Total (seg)	Media (seg)
1	0.68929577	0.00013786
2	0.68921471	0.00013784
3	0.68915367	0.00013783

### ***Execuções do programa exemplo sem variáveis alteradas, para efeito de comparação***

Filho	Total (seg)	Media (seg)
1	0,14078701	0,00014079
2	0,13958097	0,00013958
3	0,14059603	0,00014060

As duas variáveis alteraram significativamente o teste. Em média, o desvio total aumentou 0.54 segundos, porém o desvio médio se manteve o mesmo. O desvio médio não é alterado significativamente, pelo fato de ser calculado em função da variável "NO\_OF\_ITERATIONS".

**-= Resultados da execução do programa modificado -=**

**Programa com o tempo de dormência crescente**

***Execuções com tempos de dormência crescentes e sem carregamento de CPU***

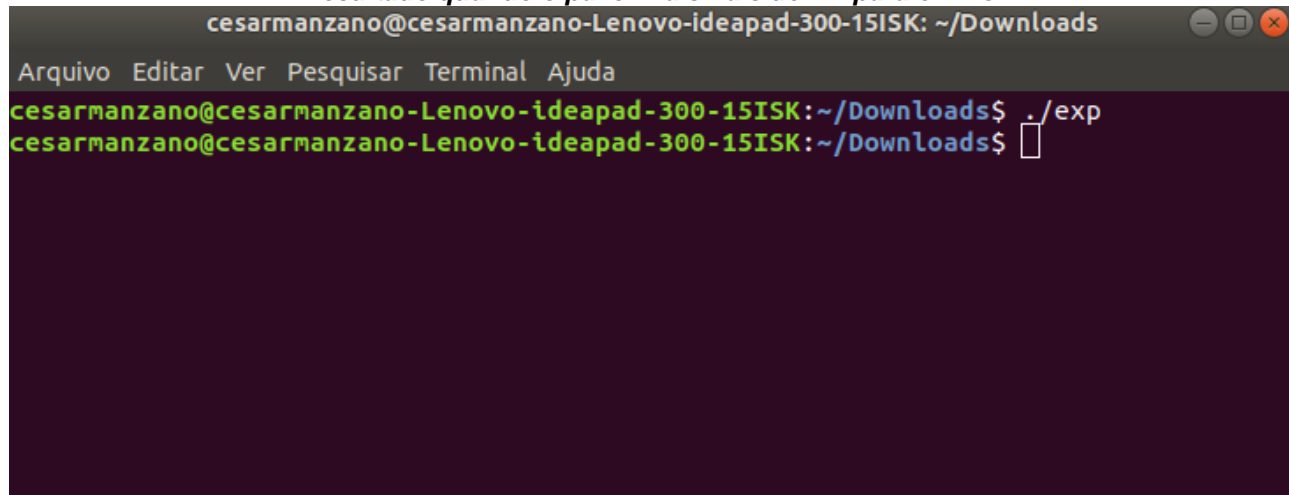
<b>Dormência</b>	<b>Filho</b>	<b>Total (seg)</b>	<b>Media (seg)</b>
400	1	0.49241501	0.00049241
	2	0.49287599	0.00049288
	3	0.49278200	0.00049278
	4	0.49278301	0.00049278
	5	0.49283099	0.00049283
600	1	0.73876202	0.00073876
	2	0.73868001	0.00073868
	3	0.73862302	0.00073862
	4	0.73862302	0.00073862
	5	0.73863900	0.00073864
800	1	0.93290699	0.00093291
	2	0.93283999	0.00093284
	3	0.93271202	0.00093271
	4	0.93002701	0.00093003
	5	0.93269402	0.00093269
1000	1	0.13839197	0.00013839
	2	0.13848901	0.00013849
	3	0.13847697	0.00013848
	4	0.13828099	0.00013828
	5	0.13837004	0.00013837
1200	1	0.34958100	0.00034958
	2	0.34936500	0.00034937
	3	0.34925103	0.00034925
	4	0.34924400	0.00034924
	5	0.34926796	0.00034927
1400	1	0.54492104	0.00054492
	2	0.54478502	0.00054478
	3	0.54469299	0.00054469
	4	0.54469299	0.00054469
	5	0.54470098	0.00054470
1600	1	0.76534998	0.00076535
	2	0.75982201	0.00075982
	3	0.76496601	0.00076497
	4	0.75978398	0.00075978
	5	0.75974298	0.00075974
1800	1	0.94939899	0.00094940
	2	0.94913805	0.00094914
	3	0.94931304	0.00094931
	4	0.94921803	0.00094922
	5	0.94900095	0.00094900
2000	1	0.16089606	0.00016090
	2	0.16076612	0.00016077
	3	0.16067195	0.00016067
	4	0.16283798	0.00016284
	5	0.16280007	0.00016280
	1	0.34491110	0.00034491
	2	0.34482098	0.00034482



2200	3	0.34487200	0.00034487
	4	0.34482098	0.00034482
	5	0.34485292	0.00034485

### Programa com o pai mandando sinais de kill para os filhos

#### *Resultado quando o pai envia sinais de kill para o filho*



```

cesarmanzano@cesarmanzano-Lenovo-ideapad-300-15ISK: ~/Downloads
Arquivo Editar Ver Pesquisar Terminal Ajuda
cesarmanzano@cesarmanzano-Lenovo-ideapad-300-15ISK:~/Downloads$ ./exp
cesarmanzano@cesarmanzano-Lenovo-ideapad-300-15ISK:~/Downloads$ 

```

Quando o pai envia um sinal de kill para cada filho, obtemos o resultado acima, no qual não aparece nada ao executarmos o programa.

## **-= Respostas das perguntas -=**

### **Perguntas do relatório**

**Pergunta 1:** *Apresente a linha de comando para compilar o programa exemplo, de tal maneira que o executável gerado receba o nome de "experimento1" (sem extensão).*

**Resposta:** gcc Experimento1.c -o experimento1.

**Pergunta 2:** *Descreva o efeito da diretiva &.*

**Resposta:** Se o usuário colocar "&" após um comando, o shell não vai esperar que ele termine e, assim envia imediatamente o caractere \$ no prompt tornando uma tarefa em background.

**Pergunta 3:** *Qual é o motivo do uso do "./"? Explique por que a linha de comando para executar o gcc não requer o "./"*

**Resposta:** - O uso do "." se deve pela necessidade de indicar qual pasta e qual arquivo dentro da pasta será executado. Sendo assim o ponto (.) é usado para indicar que o arquivo está do diretório atual e a barra (/) para mostrar qual arquivo do diretório será executado. Para o gcc não é necessário o uso do "." pois ele é um programa instalado no computador, sendo assim o Sistema Operacional entende que ao colocar gcc se trata do programa independente do arquivo que será aberto ou compilado.

**Pergunta 4:** *Apresente as características da CPU do computador usado no experimento.*

**Resposta:** Intel® Core™ i7-4790 CPU @ 3.60GHz x 8.

**Pergunta 5:** *O que significa um processo ser determinístico?*

**Resposta:** Um processo determinístico é aquele em que, dado um conjunto de entradas, sempre resultará no mesmo conjunto de saídas. Um valor de entrada do usuário ou a geração de um dado aleatório no programa, por exemplo, já não fazem o processo ser determinístico, e por isso não é comum ocorrer processos dessa natureza.

**Pergunta 6:** *Como é possível conseguir as informações de todos os processos existentes na máquina, em um determinado instante?*

**Resposta:** É possível ver os processos que estão rodando na máquina em um determinado instante pelo comando "ps ax". Para verificar qual processo é o pai e seus respectivos filhos, é necessário rodar o comando "ps ax f". Para verificar quais são os processos que estão rodando em tempo real é utilizado o comando top, permitindo uma melhor visão sobre os processos que estão rodando no computador.

### **Perguntas do programa**

**Pergunta 1:** *o que o compilador gcc faz com o arquivo .h, cujo nome aparece após o include?*

**Resposta:** Os arquivos .h são bibliotecas que, em sua maioria, estão em uma pasta de includes. Na fase de pré-processamento o compilador adiciona os arquivos .h no programa.

**Pergunta 2:** *apresentar (parcialmente) e explicar o que há em <stdio.h>.*

**Resposta:** A biblioteca <stdio.h> possui diversas funções responsáveis pela entrada e saída de dados do programas, como printf() (para a impressão de informações e dados na tela) e scanf() (para a entrada de dados no programa). A biblioteca também possui várias definições para variáveis e constantes.

**Pergunta 3:** *Qual é a função da diretiva include (linha que começa com #), com relação ao compilador?*

**Resposta:** Ao usarmos a diretiva `include`, o compilador adiciona os arquivos, bibliotecas que aparecem após a diretiva.

**Pergunta 4:** *O que são e para que servem `argc` e `argv`?*

**Resposta:** `argc` indica o número de argumentos que foram passados ao chamar o programa. Já o `argv` é um vetor que contém esses argumentos. Cada string do `argv` é um dos parâmetros da linha de comando.

**Pergunta 5:** *Qual a relação entre `SLEEP_TIME` e o desvio, nenhuma, direta ou indiretamente proporcional.*

**Resposta:** Nenhuma.

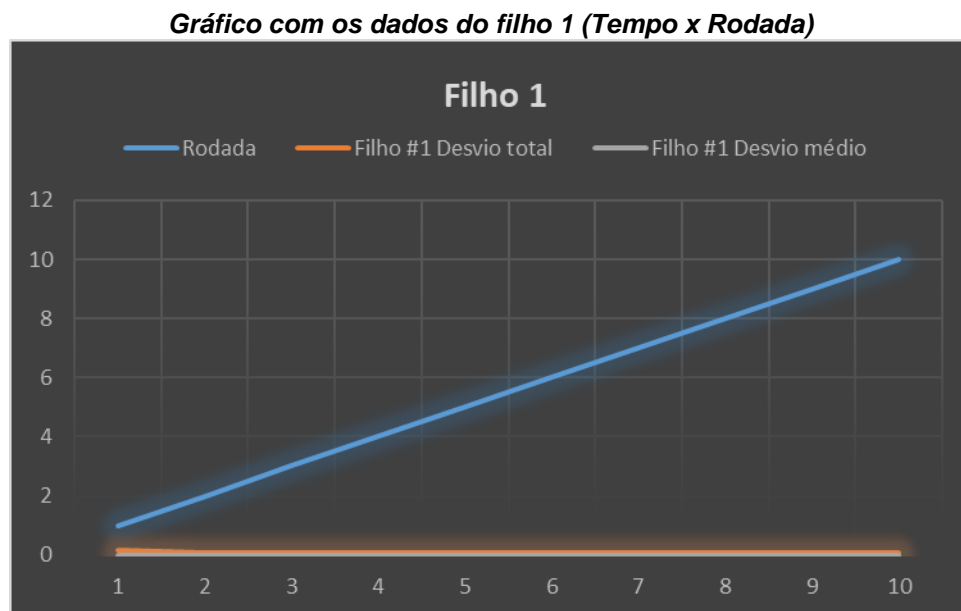
## **-= Análise dos Resultados -=**

### **Tarefa 1**

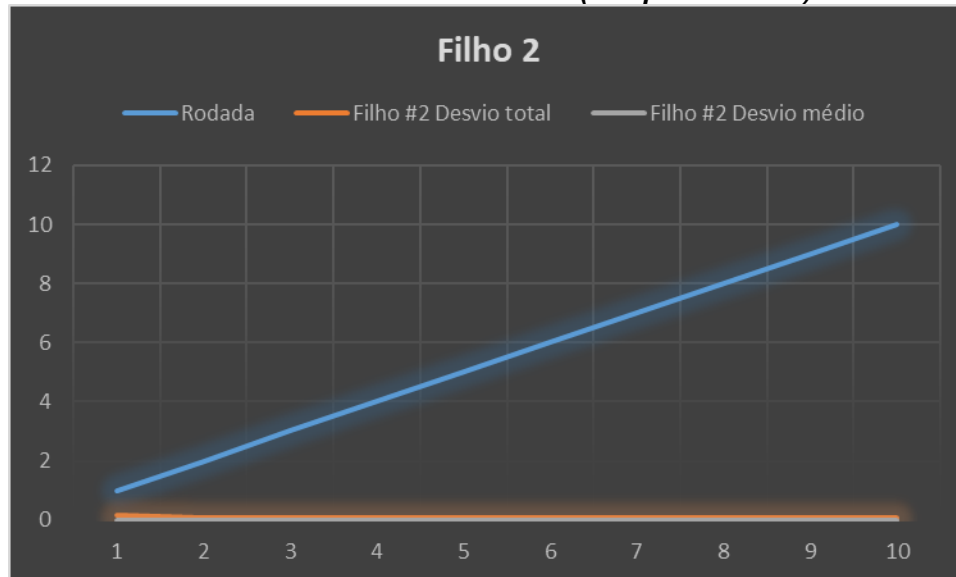
Comparando a primeira rodada de testes (sem o carregamento da CPU) com a segunda rodada (com cinco programas consumindo CPU), notamos uma diferença notável em ambos os desvios. Os desvios da segunda rodada diminuíram em relação a primeira rodada. Quando estamos com a atividade baixa da CPU (caso da primeira rodada), o processador roda em uma frequência bem menor do que ele pode. Isso pode explicar a diminuição dos desvios na segunda rodada, uma vez que a frequência do processador está maior, podendo realizar processos mais rapidamente.

Da segunda até a décima rodada não percebemos diferenças significativas nos desvios. Como os processadores atuais possuem processadores multicore, os programas que consumiam CPU poderiam estar sendo processados em diferentes núcleos, mantendo uma pequena diferença entre os desvios.

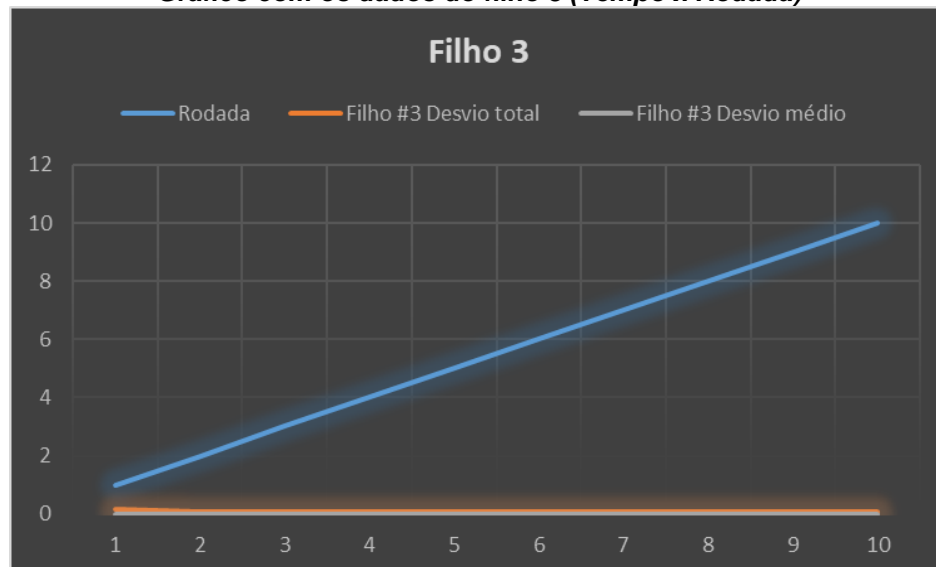
A seguir, mostra-se os gráficos dos desvios dos filhos.



**Gráfico com os dados do filho 2 (Tempo x Rodada)**



**Gráfico com os dados do filho 3 (Tempo x Rodada)**



O tempo de desvio e desvio médio são bem pequenos assim como suas variâncias, e por isso as linhas geradas no gráfico configuram praticamente uma reta, tendo o tempo de desvio médio e desvio total bem próximos, por isso uma linha sobressai a outra.

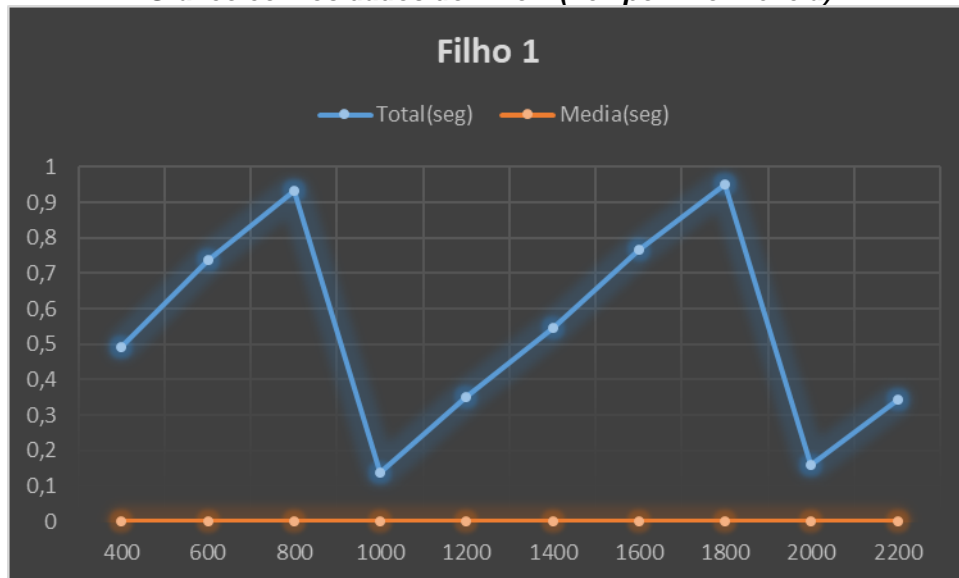
## **Tarefa 2**

### **Programa com o tempo de dormência crescente**

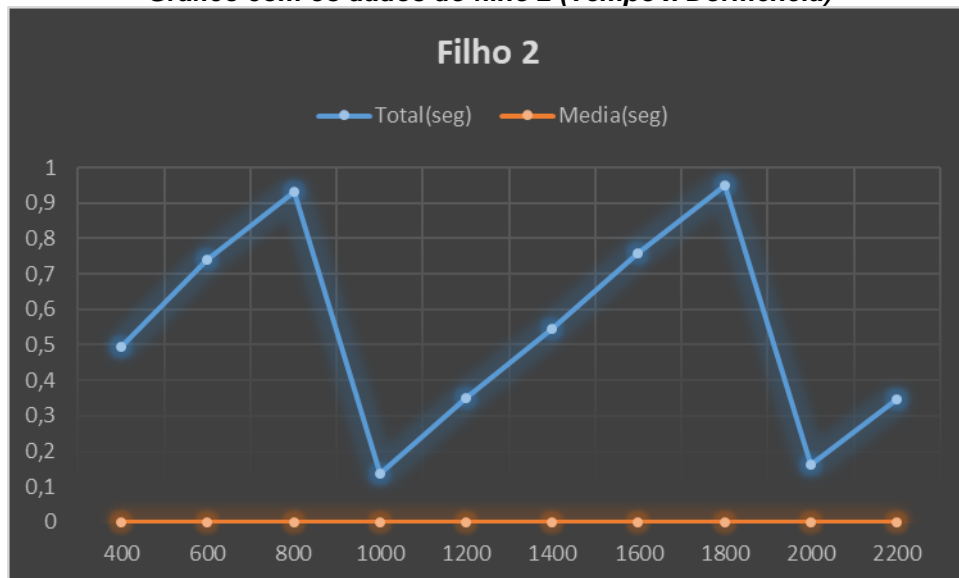
Na segunda tarefa foi proposta a modificação do programa, aumentando o número de filhos (de 3 para 5) e o tempo de “dormência” do programa para ver quais seriam as variações do sistema em diferentes cenários. O tempo de dormência foi alterado 10 vezes, sendo que a primeira rodada começaria com 400ms de tempo de dormência e as rodadas seguintes o tempo de dormência seria um múltiplo de 200ms.

Os gráficos abaixo mostram de forma individual o comportamento de cada um dos 5 filhos de acordo com cada tempo.

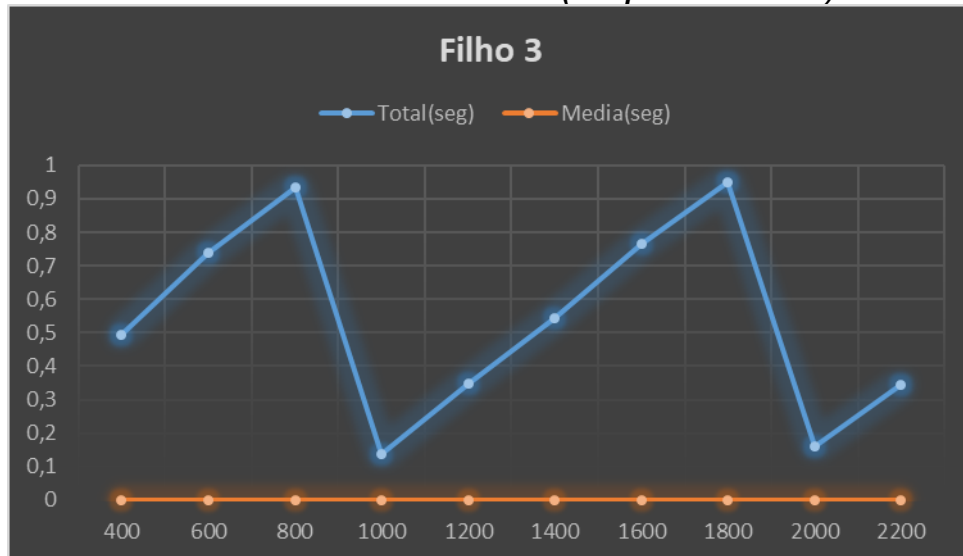
**Gráfico com os dados do filho 1 (Tempo x Dormência)**



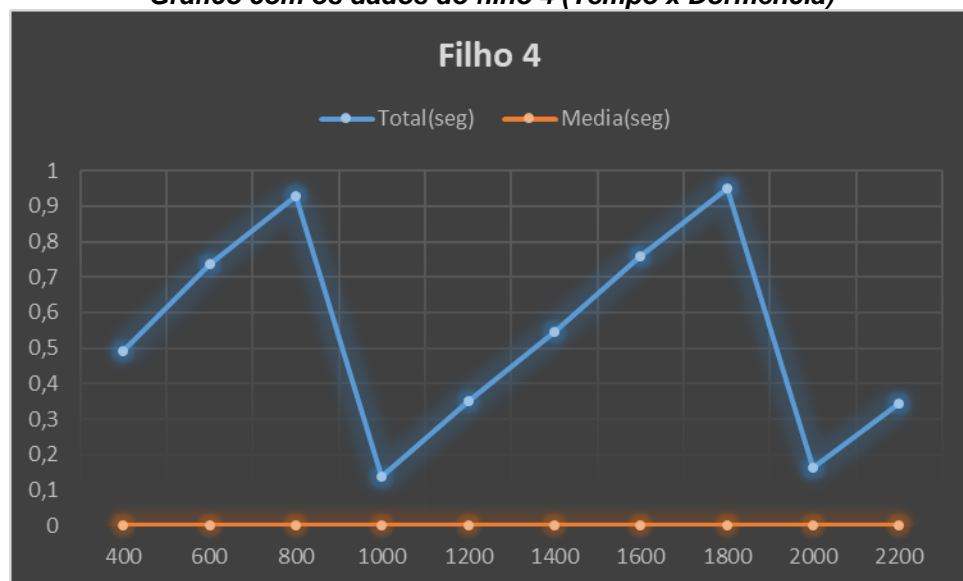
**Gráfico com os dados do filho 2 (Tempo x Dormência)**



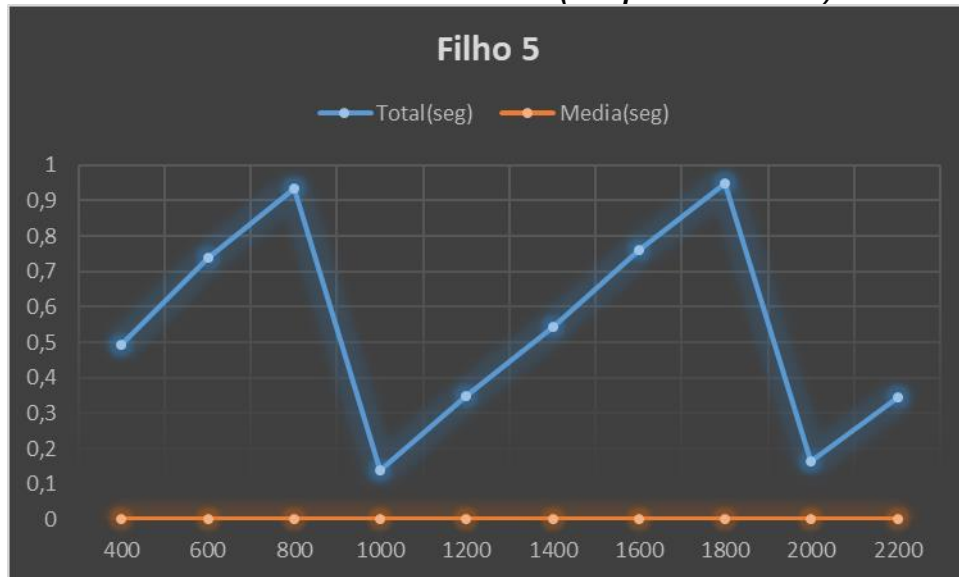
**Gráfico com os dados do filho 3 (Tempo x Dormência)**



**Gráfico com os dados do filho 4 (Tempo x Dormência)**



**Gráfico com os dados do filho 5 (Tempo x Dormência)**



Após uma análise aprofundada dos resultados obtidos ao longo dos testes, chegamos à conclusão que a CPU sofreu uma variação de frequência, o que foi determinante para os resultados obtidos ao longo do teste. Também observamos que os resultados, se comparados entre si, não tiveram quase nenhuma variação, deixando os gráficos praticamente idênticos.

#### **Programa com o pai mandando sinais de kill para os filhos**

Para mandar o sinal de kill para cada um dos filhos, armazenamos o pid de cada filho em um vetor e usamos a função `kill()`, mandando um sinal para o filho correto. Como mostrado anteriormente, o prompt de comando não exibía nada na tela. Isso ocorreu pelo fato de que o processo pai manda o sinal de kill, antes do filho executar o que deveria, no caso calcular os desvios totais e médios.



## **-= Conclusão =-**

Através deste experimento foi discutido a criação de processos e o tempo de execução. Usando o ambiente Linux como interface, dados foram coletados e analisados e estes foram explicados neste relatório. Com isso foi observado algumas variáveis que podem influenciar no tempo de processamento de tarefas concorrentes, como o número de filhos gerados, o tempo de dormência e a carga que a CPU está rodando. A CPU por sua vez tenta manter o desempenho utilizando as ferramentas disponíveis, como o aumento do clock, o uso de mais núcleos (cores) de uma CPU e o escalonamento entre processos dificultando o aumento de tempo de processamento em um mesmo programa com mais cargas, e assim mantendo uma linearidade nos gráficos aqui expostos.

Como o ambiente usado foi o Linux, também foi desenvolvido a habilidade de trabalhar melhor neste ambiente, aprendendo algumas ferramentas, linhas de comando e suas devidas finalidades, como por exemplo a execução de um “Shell Script”, que auxiliou na execução dos programas, uma vez que poderíamos rodar vários programas de uma vez. Os comandos “ps” e “top” também foram essenciais no desenvolvimento deste experimento. Estes forneciam os processos que estavam rodando, sendo que o top mostrava as informações mais especificadas, assim sendo útil principalmente na primeira tarefa, na qual tínhamos que ter a certeza que havia programas consumindo a CPU.

Também foram estudadas algumas funções como o “exec()”, que cria uma ligação entre dois programas, chamando no programa pai o programa filho, a função “fork()” que de fato gera os processos filhos, e o “wait()” que espera a execução dos filhos para então finalizar o pai.

Com todo este processo do experimento foi possível fortalecer melhor as informações adquiridas nas aulas teóricas, criar relações mais próximas entre os conceitos estudados e o funcionamento de um processador, gerando o conhecimento prático necessário para o entendimento correto dos tópicos abordados no experimento.