

## Experimento #1

### Sistemas Operacionais A

#### Introdução:

O experimento trabalha em cima da função `fork()`, que cria processos filhos por meio da duplicação de um processo pai.

No programa exemplo são gerados 3 filhos a partir do `Experimento1.c`, além dos filhos serem tratados no mesmo programa que gera o a duplicata de processos (programa de tratamento do processo pai).

Já o programa modificado não lida com o tratamento dos processos filhos, sendo essa função encarregada a outro arquivo. Para tal, utilizamos a função `exec()` que chama a execução de outro arquivo, no programa pai, deixando a ele somente a função de diferenciar os pais dos filhos e finalizar os processos filhos se desejável.

#### Apresentação Dos Erros de Sintaxe e/ou Lógica do programa Exemplo:

- **Erros de Sintaxe:**
  - Linha 44, adicionando a biblioteca `<stdlib.h>` para o uso da função de `exit()`
  - Linha 113, declarando a variável `rtn` como inteiro (`int`), e inicializando-a com 1.
  - Linha 114, alterando de `count+-` para `count++`, permitindo o funcionamento do `for()`.
- **Erros de Lógica:**
  - Linha 115, verificando se `rtn` é diferente e não igual a zero. (`if rtn !=0`)
  - Linha 176, realizando o `for()` enquanto `count` for menor que `NO_OF_CHILDREN`. (Apenas a inversão do sinal presente na função)

#### Respostas às perguntas:

- **Contidas no texto do Experimento:**
  - **1** - Linha de comando para compilar o programa: `gcc Experimento1.c -o experim1`
  - **2** - A diretiva `"&"` permite a geração de dois ou mais processos concorrentes pela CPU, com `./experim1 & ./experim1`, a CPU é disputada por dois processos que executam o `"experim1"`.
  - **3** - Ao utilizar o comando para execução do gcc estamos apenas gerando um arquivo executável referente ao programa `".C"`. Já no momento que usamos `./`, estamos pedindo para o terminal execute o arquivo executável gerado pelo gcc.
  - **4** - Características da CPU: Intel(R) Core (TM) i5-6400 CPU @ 2.70GHz

- **5 –** São processos cuja execução de um estado A sempre resulta no mesmo estado B de resposta, além disso neste processo não há variação de valores.
- **6 –** Utilizando o comando `os` que retorna os processos em execução no instante que é chamado.

- **Contidas no código fonte:**

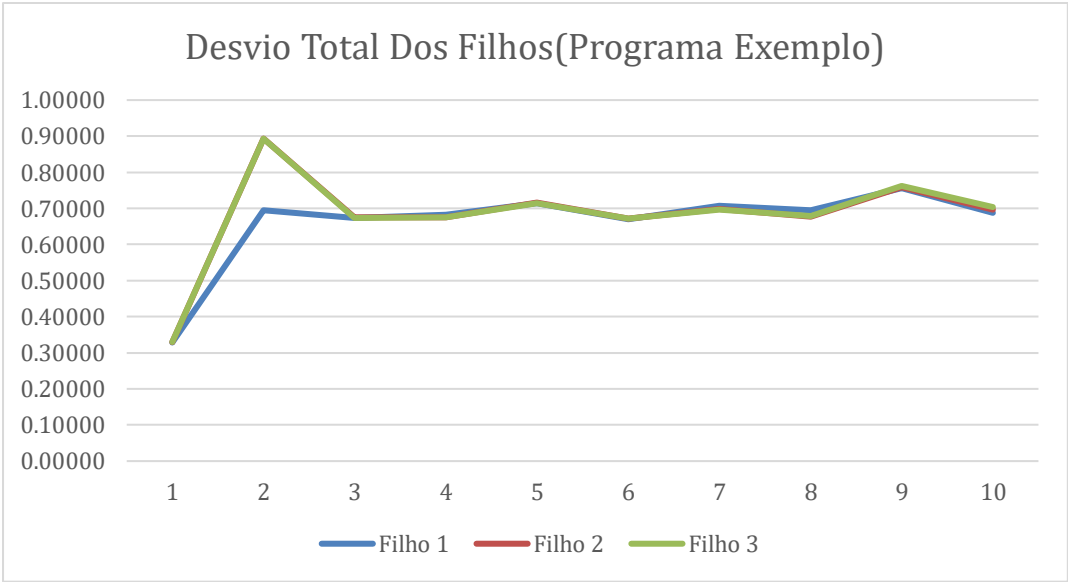
- **1 - O que o compilador gcc faz com o arquivo .h, cujo nome aparece após o include?**  
O compilador realiza a leitura do arquivo compila e assimila as funções presentes nele, para usar no programa na qual são chamadas.
- **2 - Apresentar (parcialmente) e explicar o que há em <stdio.h>**  
Nesta biblioteca estão contidas as principais funções de entrada e saída de dados em C, tal como `printf()`, e `scanf()`.
- **3 - Qual é a função da diretiva include (linha que começa com #), com relação ao compilador?**  
A diretiva permite a inclusão de diferentes bibliotecas ao programa principal, agregando diferentes funções com facilidade e rapidez.
- **4 - O que são e para que servem argc e argv? Não esqueça de considerar o \* antes de argv.**  
O `argc` representa o número de argumentos entregues à função `main`, já o `*argv[]`, é um vetor de strings composto pelos argumentos passados à `main`, começando sempre com o nome do executável do próprio arquivo.
- **5 - Qual a relação entre SLEEP\_TIME e o desvio, nenhuma, direta ou indiretamente proporcional?**  
A relação entre eles é diretamente proporcional até a mesma ser múltipla de 1000, pois quando isso ocorre gera um “reset” no desvio.

# Resultados da Execução do Programa Exemplo:

- Dados recolhidos

Rodada	Filho 1°		Filho 2°		Filho 3°		Cargas
	Desvio Total	Desvio Médio	Desvio Total	Desvio Médio	Desvio Total	Desvio Médio	
1	0,32847	0,00033	0,32945	0,00033	0,32986	0,00033	0
2	0,69540	0,00070	0,89340	0,00069	0,89321	0,00070	5
3	0,67393	0,00067	0,67556	0,00068	0,67274	0,00067	10
4	0,68284	0,00068	0,67542	0,00068	0,67568	0,00068	15
5	0,71386	0,00071	0,71531	0,00072	0,71411	0,00071	20
6	0,66971	0,00067	0,67239	0,00067	0,67103	0,00067	25
7	0,70673	0,00071	0,69891	0,00070	0,69596	0,00070	30
8	0,69501	0,00070	0,67629	0,00068	0,67857	0,00070	35
9	0,75603	0,00076	0,75782	0,00076	0,76174	0,00076	40
10	0,68736	0,00069	0,69586	0,00070	0,70361	0,00070	45

- Gráfico



## Resultados da Execução do Arquivo Modificado:

- **Dados Recolhidos**

- **Filho 1**

Rodada	Filho 1°	
	Desvio Total	Desvio Médio
1	0,48475	0,00048
2	0,72564	0,00073
3	0,95811	0,00096
4	0,27833	0,00028
5	0,66493	0,00066
6	0,86268	0,00086
7	0,95437	0,00095
8	0,98694	0,00099
9	0,30030	0,00030
10	0,57556	0,00058

- **Filho 2**

Rodada	Filho 2°	
	Desvio Total	Desvio Médio
1	0,48505	0,00049
2	0,72529	0,00073
3	0,95820	0,00096
4	0,27662	0,00028
5	0,66598	0,00067
6	0,86178	0,00086
7	0,95478	0,00095
8	0,98727	0,00099
9	0,29891	0,00030
10	0,57717	0,00058

- **Filho 3**

Rodada	Filho 3°	
	Desvio Total	Desvio Médio
1	0,48484	0,00048
2	0,72377	0,00072
3	0,95843	0,00096
4	0,27722	0,00028
5	0,66516	0,00067
6	0,86218	0,00086
7	0,95497	0,00095
8	0,98657	0,00099
9	0,29791	0,00030
10	0,57649	0,00058

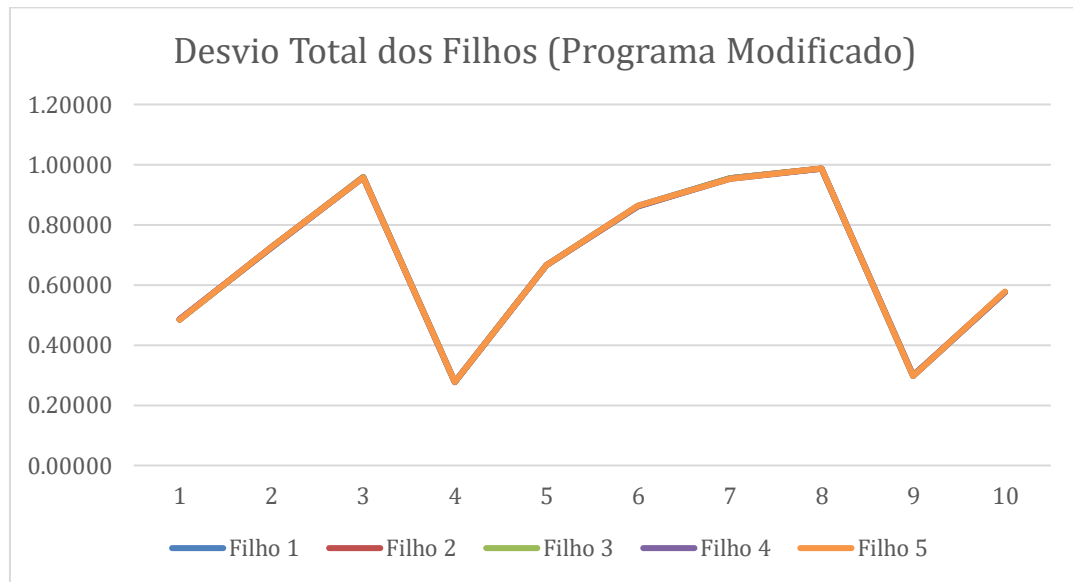
○ **Filho 4**

Rodada	Filho 4°	
	Desvio Total	Desvio Médio
1	0,48601	0,00049
2	0,72398	0,00072
3	0,95768	0,00096
4	0,27725	0,00028
5	0,66499	0,00066
6	0,86134	0,00086
7	0,95342	0,00095
8	0,98653	0,00099
9	0,29818	0,00030
10	0,5766	0,00058

○ **Filho 5**

Rodada	Filho 5°	
	Desvio Total	Desvio Médio
1	0,48451	0,00048
2	0,7245	0,00072
3	0,95789	0,00096
4	0,27767	0,00028
5	0,66621	0,00067
6	0,86241	0,00086
7	0,95362	0,00095
8	0,98752	0,00099
9	0,29787	0,00030
10	0,57783	0,00058

- **Gráfico**



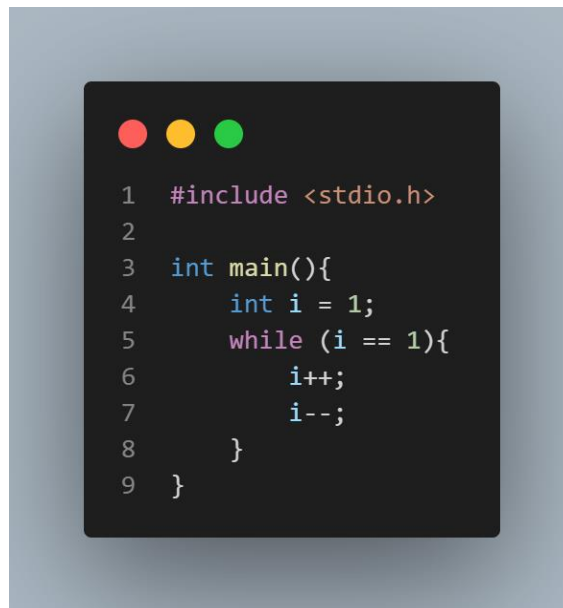
### **Análise dos resultados:**

Olhando para os dados de execução do programa exemplo pode-se notar uma consistência nos resultados a partir da segunda, até a décima rodada, mantendo seu desvio em torno de 0,7 ademais das cargas adicionadas.

Já visando os dados do programa modificado percebe-se uma variação uniforme de crescimento, sempre resultando resetando desvio para por conta do aumento do tempo de dormência da CPU sempre que a mesma for múltipla de 1000 (comprovando a resposta da questão 5 contida no código fonte do experimento).

Com isso, a CPU apresentou maior eficiência com os testes realizados pelo programa exemplo que, apesar das cargas adicionadas, apresentou desvios máximos menores que os do programa modificado.

## Programa Usado para Aumento de Carga:



```
1  #include <stdio.h>
2
3  int main(){
4      int i = 1;
5      while (i == 1){
6          i++;
7          i--;
8      }
9  }
```

## Conclusão:

Após a realização dos testes e análise dos resultados, foi agregado, por meio do experimento, uma visão mais palpável do impacto do Muulti-Tasking realizado pela CPU, provando que apesar dos demais processos concorrentes manteve sua consistência em todas as rodadas sendo próximo da eficiência do programa modificado.