**Experimento 1**

**Sistemas Operacionais A**

**-= Integrantes =-**

1. Beatriz Morellato Lorente RA:18071597

2. Cesar Marrote Manzano RA: 18051755

3. Fabrício Silva Cardos RA: 18023481

4. Pedro Ignácio Trevisan RA: 18016568

**-= 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 realizado refere-se à duração da execução de um trecho de programa dentro do computador, que nunca pode ser medido com certeza. A percepção de tempo dentro de um ambiente computacional é algo impreciso, devido à interferência de um conjunto grande de variáveis.

O experimento deve permitir perceber como duas das variáveis existentes em um sistema computacional impactam o tempo de duração de um processo. O experimento também permite conhecer como se realiza a criação de processos em um ambiente multitarefa. Para isso, usa-se um programa exemplo, cuja execução deve ser acompanhada por diferentes cargas de máquina. Usa-se também um programa modificado, baseado no exemplo, que deve ser executado de maneira a se ter diferentes tempos de auto-bloqueio do processo.

O experimento também permite o contato com um número grande de funções Unix responsáveis pela manipulação de processos e tomada de tempo.

**-= 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 quem ao colocar gcc se trata do programa independente do arquivo que será aberto ou compilado.

**Perguntas do programa**

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

Resposta:

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

Resposta: A bilbioteca <stdio.h> possui diversas funções reponsáveis 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:

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

Ao compilar o código foi possível perceber alguns erros de sintaxe e lógica do programa. Os problemas estão listados abaixo, seguidos de suas soluções

**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.

Programa corrigido:

int 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)

**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 perceber a diferença entre 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);

**-= 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 | 0.6812000 | 0.00006812 | 0.06951094 | 0.00006951 |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Filho/Carga** | **Total(seg)** | **Media(seg)** |
| 1/0 | 0,14078701 | 0,00014079 |
| 2/0 | 0,13958097 | 0,00013958 |
| 3/0 | 0,14059603 | 0,00014060 |
| 1/5 | 0,06702602 | 0.00006703 |
| 2/5 | 0.6812000 | 0.00006812 |
| 3/5 | 0.06951094 | 0.00006951 |
| 1/10 |  |  |
| 2/10 |  |  |
| 3/10 |  |  |
| 1/15 |  |  |
| 2/15 |  |  |
| 3/15 |  |  |
| 1/20 |  |  |
| 2/20 |  |  |
| 3/20 |  |  |
| 1/25 |  |  |
| 2/25 |  |  |
| 3/25 |  |  |
| 1/30 |  |  |
| 2/30 |  |  |
| 3/30 |  |  |
| 1/35 |  |  |
| 2/35 |  |  |
| 3/35 |  |  |
| 1/40 |  |  |
| 2/40 |  |  |
| 3/40 |  |  |
| 1/45 |  |  |
| 2/45 |  |  |
| 3/45 |  |  |

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

#include<stdio.h>

int main ()

{

int i = 0;

while (1)

{

i++;

}

return 0;

}

Este programa foi executado a partir da segunda tomada de tempos um número de vezes igual à rodada menos um. Por exemplo, na quinta rodada, existiam quatro instâncias concorrendo com o exemplo e na décima, nove.

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

***Execuções com tempos de “dormência” crescentes e sem carregamento de CPU (em segundos)***

|  |  |  |  |
| --- | --- | --- | --- |
| Tempo dormência | Tempo Obtido | Tempo Esperado | Desvio Tempo |
| 40 |  |  |  |
| 40 |  |  |  |
| 80 |  |  |  |
| 80 |  |  |  |
| 160 |  |  |  |
| 160 |  |  |  |
| 320 |  |  |  |
| 320 |  |  |  |
| 640 |  |  |  |
| 640 |  |  |  |

**-= Análise dos Resultados =-**

Observando os resultados das dez execuções com o programa exemplo, verifica-se que o valor aproximado do desvio médio de 20 mili segundos, para cada uma das 500 interações que deveria demorar em torno de 20 micro segundos de dormência mais a execução das instruções no “loop”, aponta para um tempo relativamente alto de demora para que o processo chame a rotina de dormência, fique bloqueado, durma, seja desbloqueado e volte a ser executado para a segunda tomada de tempo. É importante lembrar que a própria tomada de tempo consome CPU e leva tempo.

Quanto ao consumo de CPU pelos irmãos, nota-se que estes, uma vez que assumem a CPU, rapidamente a liberam pois solicitam dormência.

Quanto ao consumo de CPU pelas instâncias que deveriam carregar a máquina, nota-se que houve uma leve tendência de aumento do desvio. Porém, reconhece-se que o número de instâncias do programa para carga do sistema deveria ter sido bem maior que as quantidades escolhidas.

Percebe-se nas rodadas 5 e 8 um aumento do desvio, maior inclusive do que o da rodada 10, possivelmente ocasionado pela atividade do sistema operacional. Algumas rotinas devem ter contribuído para concorrer pela CPU.

No programa alterado, também se nota que o desvio total obtido, de aproximadamente dez segundos, considerando as 500 interações em cada uma das 10 rodadas, também aponta para um desvio médio de 20 mili segundos, que é afetado pelo tempo de dormência. Esses 20 mili segundos por interação correspondem ao tempo para que o processo pai chama a rotina de dormência, fica bloqueado, dorme, é desbloqueado e volta a ser executado para a segunda tomada de tempo.

Como os filhos continuam também a dormir, na mesma proporção do pai, com o aumento da dormência, o período em que o processo se encontra bloqueado deve ser suficiente para a execução dos outros processos, assim, após o desbloqueio, o tempo de espera pela CPU parece diminuir, ocasionando um declínio nos valores do desvio. Neste caso na ordem do tempo esperado de dormência. Aqui, novamente, reconhece-se que poderia haver escolha de tempos de dormência melhores para que o declínio ficasse mais acentuado.

Comparando os valores de desvio obtidos com as execuções do programa exemplo e aqueles obtidos com o programa modificado, nota-se que com uma carga pequena do sistema, frente ao um tempo de dormência constante, o desvio quase permanece constante. Enquanto que, com a variação do tempo de dormência para maior e manutenção da carga de máquina, observa-se a diminuição do desvio na ordem do tempo de dormência.

**-= Conclusão =-**

Com o experimento foi possível atingir os objetivos do experimento. Quais sejam, perceber como realizar um experimento usando o compilador gcc em ambiente Linux, coletar dados, analisá-los e gerar um relatório. Foi possível também perceber como ocorre a criação de processos no Linux, como se pode realizar a medição da duração da execução de um programa ou de parte dele. Finalmente, foi possível perceber o conceito de multitarefa e o que este envolve, i.e., como uma máquina com uma CPU realiza a execução concorrente dos processos e como essa concorrência impacta o tempo de execução de um dado processo.

Notou-se que entre as variáveis que impactam o tempo de execução de um processo encontram-se: o número e a natureza dos processos que estejam concorrendo pela CPU; além do número e a duração dos bloqueios a que é submetido o processo. Notou-se também que a própria tomada de tempo pode sofrer desvios, por se tratar da chamada a uma rotina do sistema operacional.

Finalmente, é interessante comentar que esta oportunidade de estar realizando este tipo de experimento está servindo para consolidar muitas ideias que na teoria ficam meio que abstratas e. também, claro, de corrigir outras.