Sistemas Paralelos e Distribuídos

Práticas – Aula 2

Revisão

- Medição de tempo de execução
- Avaliação de carga
- Automatização
- Duração de chamadas a funções

Tempo de execução

No shell:

Usamos o comando time

#> time md5dum /var/log/syslog
#>time dd if=/dev/zero of=/dev/null bs=5G count=10

Em C:

Usamos as funções:

gettimeofday( )
clock_gettime( )

Tempo de execução

clock_gettime

Incluir a biblioteca ***time.h*** e utilizar a struct ***timespec***

tv_sec – tempo em segundos
tv_nsec – tempo em nanosegundos

Medir o tempo
clock_gettime(CLOCK_MONOTONIC, &ts)

Obter o tempo atual
clock_gettime(CLOCK_REALTIME, &ts)

Tempo de execução (exemplo)

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

int main() {
    struct timespec inicio, fim;

    clock_gettime(CLOCK_MONOTONIC, &inicio);

    for (int i = 0; i < 100000000; i++);

    clock_gettime(CLOCK_MONOTONIC, &fim);

    double percorrido = (fim.tv_sec - inicio.tv_sec) + (fim.tv_nsec - inicio.tv_nsec) / 1e9;
    printf("O tempo de execucao eh: %.9f segundos\n", percorrido);

    return 0;
}
```

Tempo de execução de uma chamada do sistema

Exemplo de chamada ao sistema

https://medium.com/@noransaber685/understanding-important-system-calls-in-c-fork-open-read-close-chdir-getline-and-access-2836cc761b17

```
#include <unistd.h>

int main()
{

	int resultado = access("/var/log/nada",R_OK);
	return resultado;
}
```

Exercício: Realizar a medição do tempo de uma chamada de sistema

Utilizando a chamada ***access()*** (ou qualquer outra chamada de sistema de sua preferência), implemente um programa que mede a duração dessa chamada, tal como fora realizado no exemplo com o ***loop for***.

Tempo de uma chamada de sistema: Resolução

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

int main() {
    struct timespec inicio, fim;

    clock_gettime(CLOCK_MONOTONIC, &inicio);

     int resultado = access("/var/log/syslog",X_OK);

    clock_gettime(CLOCK_MONOTONIC, &fim);

    long percorrido = (fim.tv_sec - inicio.tv_sec) * 1000000000L + (fim.tv_nsec - inicio.tv_nsec);
    printf("O tempo de execucao eh: %.ld nanosegundos\n", percorrido);

    return resultado;
}
```

Medindo o tempo para criação e término de processos

Conhecendo as funções para criação de processo ***- fork()*** - façamos um programa que irá criar um número alto de processos e tenhamos as medições de duração entre a criação e o fechamento (***fork*** e ***wait***). Calculemos a média de tempo para encerramento dos processemos e tenhamos em conta os números máximos para não sobrecarregarmos o sistema.

#> ulimit -a

Medindo execução de processos: Resolução

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
#include <time.h>
#define NUM_PROC 60000

int main()
{
   pid_t cpid;
   struct timespec inicio, fim;
   double total = 0.0;

   for (int i=0;i<NUM_PROC;i++)
   {
          clock_gettime(CLOCK_MONOTONIC, &inicio);
          pid_t pid = fork();
          if (pid  < 0)
          {
                perror("Fork failed");
                exit(EXIT_FAILURE);
          }
          else if (pid  == 0)
               {
                 _exit(0);
               }
               else
               {
                     waitpid(pid,NULL,0);
                     clock_gettime(CLOCK_MONOTONIC, &fim);
                     double percorrido = (fim.tv_sec - inicio.tv_sec) * 1000000000L + (fim.tv_nsec - inicio.tv_nsec);
                     printf("O tempo para o wait eh de: %.fnanosegundos\n", percorrido);
                     total += percorrido;
               }
   }
   printf("Tempo medio do fork-wait: %.3f ns\n", total / NUM_PROC);
   return 0;
}
```

Medindo o tempo para criação e término de processos: Automatismos

Faça o cálculo da média de tempo, com e sem automatismo shell

#> ./many-forks | awk -F" " {'SUM=SUM+$8'}END{'print SUM/NR'}

Compare ambos

#>./many-forks > /tmp/output ; cat /tmp/output | awk -F" " {'SUM=SUM+$8'}END{'print SUM/NR'} && cat /tmp/output |grep medio

Avalie o tempo de execução em diferente contextos de carga

Meça a duração em diferentes contextos do sistema. Introduza carga no sistema:

#> dd if=/dev/zero of=/dev/null | dd if=/dev/zero of=/dev/null | dd if=/dev/zero of=/dev/null | dd if=/dev/zero of=/dev/null & read ; killall dd

Utilize múltiplos terminais em paralelo e observe algum indicadores:

#> vmstat -n -w 1

#>top

#>watch -n 1 uptime

Medindo o tempo para chamada a funções

Utilize a função de ***clockgettime()*** para medir a chamada de uma função sem argumentos, versus uma segunda função, com 10 argumentos (e.g. 10 argumentos do tipo ***int***).

Resolução:
Medindo o tempo para chamada a funções

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

int com_args(int a, int aa, int aaa, int aaaa, int aaaaa, int aaaaaa, int aaaaaaa, int aaaaaaaa, int aaaaaaaaa, int aaaaaaaaaa)
{ return 0; }

int sem_args()
{ return 13; }

int main() {
    int result;
    struct timespec inicio, fim;
    double tempo_com_args, tempo_sem_args;
    double tempototal_com_args, tempototal_sem_args;

    for (int i=0;i<1000000;i++)
    {
    clock_gettime(CLOCK_MONOTONIC, &inicio);
    sem_args();  // Call function
    clock_gettime(CLOCK_MONOTONIC, &fim);
    tempo_sem_args = (fim.tv_sec - inicio.tv_sec) * 1e6 + (fim.tv_nsec - inicio.tv_nsec) / 1e3;
    tempototal_sem_args += tempo_sem_args;
    }
    tempototal_sem_args = tempototal_sem_args/1000000;

    for (int i=0;i<1000000;i++)
    {
    clock_gettime(CLOCK_MONOTONIC, &inicio);
    com_args(1,2,3,4,5,6,7,8,9,10);  // Call function
    clock_gettime(CLOCK_MONOTONIC, &fim);
    tempo_com_args = (fim.tv_sec - inicio.tv_sec) * 1e6 + (fim.tv_nsec - inicio.tv_nsec) / 1e3;
    tempototal_com_args += tempo_com_args;
    }
    tempototal_com_args = tempototal_com_args/1000000;

    printf("Tempo para chamar funcao com 10 argumentos: %.3f µs\n", tempototal_com_args);
    printf("Tempo para chamar funcao sem argumentos: %.3f µs\n", tempototal_sem_args);

    return 0;
}
```

Medindo o tempo para chamada a funções. Ressalvas!

A fim de evitar:

- Pré-carregamento de cache
- Grande variação e erro
- Optimização do compilador

Realizamos a chamada por 1 milhão de vezes e calculamos a média.
Ao utilizarmos funções void (sem corpo) o compilador tentará optimizá-las, tornando os tempos de execução ligeiramente similares. Por isso usamos funções com retorno int.
Usamos também uma precisão de milissegundos, mas poderíamos também recorrer à escala de nanosegundos.

Conclusão:

Conciliar e utilizar todas ferramentas e conceitos abordados nestas revisões:

- Makefile
- Shell script
- Syscalls
- Gettime
- I/O Redirection
- Average calculation
- Threading and Forking

de forma a possibilitar o bom desenvolvimento do trabalho que será parte da avaliação de curso (SPD - práticas)

It is over now