E-fólio B | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Sistemas Operativos

CÓDIGO: 21111

DOCENTE: Paulo Shirley, Gracinha Carvalho, José Coelho

A preencher pelo estudante

NOME: Ivo Vieira Baptista

N.º DE ESTUDANTE: 2100927

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 16 de Maio de 2022

TRABALHO / RESOLUÇÃO:

No programa declaro variáveis globais e locais, e também crio o **array** de **mutex**, defino também uma estrutura para os números manipulados no programa.

```
/*************************************************************
    Variáveis globais
*************************************************************/
//double soma(double x[], int n); /* funcao que soma os elementos de um vector */
void *tarefa(void *arg);        /* prototipo funcao tarefa */
//etypedef struct {double *x; int n,i;} trfarg_t;  /* tipo do argumento tarefas */
pthread_mutex_t mtx_num_erros;                      /* mutex */
pthread_mutex_t *mtx_processada;
int total_de_linhas, total_de_erros;

//-----------------------------------------------------
//      definição da struct
//-----------------------------------------------------
typedef struct Numeros{
   int numero1, numero2, esperado, resultado, processada;
} Numeros;

Numeros numeros[1000];
```

O programa foi inspirado no Laboratório 3 onde aprendemos sobre a compilação e execução de pequenos programas, que efetuam chamadas a funções do sistema operativo UNIX/Linux, para programação multitarefa.

Nos exemplos do Laboratório 3 estudamos o programa **pth09.c** em que a programação é feita de modo ao troço de código correspondente á região critica entre o par de chamadas a **pthread_mutex_lock()** e **pthread_mutex_unlock()** seja o mais breve possível, de modo ao programa no geral ser mais eficiente, minimizando o tempo de espera das outras tarefas.

Também foi nos fornecida uma função combinações que retorna e lee números inteiros, mas raparei que no ficheiro casos.txt fornecido, tem duas linhas com números que contem decimais, na linha 840 e 919 especificamente, era mais simples de ler o ficheiro com o comando **fscanf** diretamente em inteiros, mas utilizei a o **fgets** no nosso ciclo **while** para obter os dados em **string** e logo converter com **strtok** e **atoi**, para valores inteiros, sendo esta a tarefa principal, testei a memoria se foi alocada corretamente com o comando **malloc**, aqui é onde existe realmente o **array** na memoria:

```
/* Testando se memoria foi alocada correctamente */
mtx_processada = malloc(total_de_linhas * sizeof(pthread_mutex_t));
if( mtx_processada == (pthread_mutex_t *) NULL) {
  perror("Erro alocando o array de mutexes");
  return 1;
}
```

Também temos uma verificação de argumentos para as tarefas entre 1-15, 20 e 30 e o ficheiro casos.txt

A forma de executar o programa por exemplo seria:

./pteste 15 casos.txt

criei vários ciclos **for** um deles para inicializar o **array** de **mutex**, outro para as estruturas de dados que servem de argumento das tarefas, validei o erro na criação da tarefa, depois esperamos que as tarefas criadas terminem, destruímos o **mutex** e liberamos a memoria alocada para todo o **array**.

No final mostro os resultados esperados com mais um **for** e um **if** que decide os casos com erro na execução do programa.

```
→  Testes ./pteste 15 casos.txt
Resultados:
Sucesso: 912
Falhas: 86
Total casos: 998
Erro no caso 405: input (30,15) esperado/observado:155117520/-131213633
Erro no caso 465: input (31,13) esperado/observado:206253075/-124129024
Erro no caso 493: input (33,14) esperado/observado:818809200/40051392
Erro no caso 512: input (35,18) esperado/observado:242600354/79433710
Erro no caso 547: input (35,15) esperado/observado:-1047024136/-28345986
Erro no caso 559: input (34,13) esperado/observado:927983760/-8098854
Erro no caso 564: input (37,15) esperado/observado:774265168/-7639118
Erro no caso 588: input (36,14) esperado/observado:-498670096/-3096940
Erro no caso 592: input (38,27) esperado/observado:1203322288/55409176
Erro no caso 604: input (39,11) esperado/observado:1676056044/153294910
Erro no caso 611: input (36,10) esperado/observado:254186856/-175309873
Erro no caso 630: input (39,19) esperado/observado:203787674/3500118
Erro no caso 642: input (40,24) esperado/observado:-1572407790/-81335683
Erro no caso 645: input (33,16) esperado/observado:1166803110/85391703
Erro no caso 650: input (42,22) esperado/observado:-1604468100/26091787
Erro no caso 653: input (31,17) esperado/observado:265182525/28327710
Erro no caso 654: input (36,15) esperado/observado:1272935264/-48593124
Erro no caso 656: input (36,21) esperado/observado:1272935264/99623393
Erro no caso 658: input (33,21) esperado/observado:354817320/-80599791
Erro no caso 668: input (35,25) esperado/observado:183579396/-66960362
Erro no caso 679: input (31,16) esperado/observado:300540195/14209042
Erro no caso 682: input (33,14) esperado/observado:818809200/40051392
Erro no caso 692: input (37,15) esperado/observado:774265168/-7639118
Erro no caso 695: input (36,10) esperado/observado:254186856/-175309873
Erro no caso 703: input (43,9) esperado/observado:563921995/86703406
Erro no caso 707: input (44,34) esperado/observado:-1813710518/-60346575
Erro no caso 728: input (38,27) esperado/observado:1203322288/55409176
Erro no caso 746: input (47,17) esperado/observado:999740566/-23390067
```

Nas funções de tarefas:

for(i=0; i< total_de_linhas; i++) {

Aqui temos acessos a uma variável **i**, que não precisa ser protegido por **mutex**, por ser esta uma variável local a cada tarefa (cada qual tem sua cópia).

Além disso, temos uma comparação dela com uma variável global. Mas essa também não precisa de proteção, já que nem a thread principal, que contém a **main(),** nem as **subthreads** alteram seu valor depois de sua inicialização, ao final do **while** da **main** que leu o conteúdo do arquivo de entrada.

```
pthread_mutex_lock(mtx_processada+i);

if( numeros[i].processada ) { // protegemos a i-ésima posição do array numeros.

   pthread_mutex_unlock(mtx_processada+i);

   continue;

} else {

   numeros[i].processada=1;

   pthread_mutex_unlock(mtx_processada+i);

}
```

Aqui temos um acesso, devidamente protegido por **lock/unlock** num conjunto de **mutexes** presente num **array**, a um campo de uma **struct** que, por sua vez, é um elemento de um **array**. A necessidade dessa proteção surge do fato do **array** ser global, e do valor desse campo poder ser alterado por qualquer das **subtarefas** em execução.

```
numeros[i].resultado = Combinacoes(numeros[i].numero1, numeros[i].numero2);

// Chamada da função Combinacoes(n,r)
```

Nessa linha há 3 acessos a espaços de memória globais, todos na mesma posição do **array** numeros (numeros[i]), mas em campos distintos da **struct**.

Os acessos aos campos numero1 e numero2 caem no mesmo caso do acesso à variável global **total_de_linhas**: não são alterados depois do **loop** da **main()** que leu o arquivo de casos.

Já o acesso ao campo resultado, em princípio, deveria ser protegido por um **mutex**, já que é uma atribuição a uma área de memória numa região global, feita pelas tarefas. No entanto, podemos constatar que essa proteção não é absolutamente necessária, já que cada tarefa testa o campo processado, antes de seguir com o processamento. Se aquela posição do **array** já tiver sido processada anteriormente por outra tarefa, a tarefa corrente vai seguir para a próxima iteração do **loop**, e não tentará fazer a atribuição. Dessa forma, essa atribuição já conta com proteção suficiente para que não seja necessário o emprego de um **mutex**.

A mesma lógica se aplica ao teste feito no **if** logo a seguir:

```
if( numeros[i].esperado != numeros[i].resultado ) {
```

No entanto, ao incrementarmos a variável contendo o total de erros encontrados, precisamos de um **mutex**, pois ela não se refere apenas àquela posição do **array**:

```
pthread_mutex_lock(&mtx_num_erros);

total_de_erros++;

pthread_mutex_unlock(&mtx_num_erros);
```