‒ Projeto Sistemas Operacionais ‒

* *Tema:* núcleo multiprogramado
* *Alunos:*

- Dante Martini Chiarelli Ramacciotti | RA 221023259

- Manuele Sabatini Christófalo | RA 221026291

- Paulo Henrique Dionysio | RA 221026169

Sumário

**EXERCÍCIOS1**

TicTac2

TicTac Modificado2

Escalonador de Corotinas2

**NÚCLEO4**

Algoritmo5

Estrutura de Dados5

Casos de Teste5

‒ Exercícios ‒

1. **TicTac**
   1. **Detalhamento do algoritmo**
   2. **Caso de teste**
2. **TicTac Modificado** 
   1. **Detalhamento do algoritmo**
   2. **Caso de teste**
3. **Escalonador de Corotinas**
   1. **Detalhamento do algoritmo**
   2. **Caso de teste**

‒ Núcleo ‒

1. **Algoritmo**

Como o código está comentado linha a linha, explicitando o que cada função e definição faz, acreditamos não haver necessidade de expansão na explicação. Portanto, segue o código comentado:

/\* SISTEMAS OPERACIONAIS II - NUCLEO MULTIPROGRAMADO =====================================

    Autores:

        - Manuele S. Christofalo | 221026291

        - Dante Ramacciotti | 221023259

        - Paulo Henrique Dionysio | 221026169

==========================================================================================\*/

// I - VARIAVEIS GLOBAIS E DEFINICOES ====================================================

#include <system.h>

// Variaveis globais ---------------------------------------------------------------

*PTR\_DESC\_PROC* prim;         //Cabeca de fila para processos prontos

PTR\_DESC d\_esc;             //Ponteiro para o descritor do escalonador

// Registradores da regiao critica -------------------------------------------------

typedef *struct* *registros*{

*unsigned* bx1, es1;      //bx = registrador base; es = registrador de segmento extra

}*regis*;

// Union dos registradores da regiao critica

typedef *union* *k*{

*regis* x;                //Registradores bx + es

*char* far \*y;            //Valor da flag de servicos

}*APONTA\_REG\_CRIT*;

*APONTA\_REG\_CRIT* a;

// Descritor de processos (BCP) ----------------------------------------------------

typedef *struct* *desc\_p*{

*char* nome[35];                                  //Nome do processo

*enum*{ativo, bloqueado, terminado} estado;       //Estado do processo

    PTR\_DESC contexto;                              //Ponteiro para descritor de contexto

*struct* *desc\_p* \*fila\_sem;                        //Fila de processos bloqueados por um semaforo

*struct* *desc\_p* \*prox\_desc;                       //Ponteiro para o proximo descritor

} *DESCRITOR\_PROC*;

typedef *DESCRITOR\_PROC* \**PTR\_DESC\_PROC*;              //Ponteiro para o descritor

// Tipo semaforo -------------------------------------------------------------------

typedef *struct*{

*int* s;                  //Parte inteira

*PTR\_DESC\_PROC* Q;        //Fila bloqueados

} *semaforo*;

// II - FUNCOES BASICAS DO NUCLEO ========================================================

// 1. Cricao do processo -----------------------------------------------------------

*void* far cria\_processo(*void* far (\*p\_address)(), *char* nome\_p[16]){

    // Definicoes base

    PTR\_DESC\_PROC descritor = (PTR\_DESC\_PROC)malloc(sizeof(struct desc\_p));     //Criacao dinamica do descritor

    strcpy(descritor->nome, nome\_p);                                            //Copia nome

    descritor->estado = ativo;                                                  //Marca o estado como "ativo"

    descritor->contexto = cria\_desc();                                          //Cria descritor de contexto

    newprocess(p\_address, descritor->contexto);                                 //Inicia descritor de contexto

    descritor->fila\_sem = NULL;                                                 //Nenhum processo bloqueado

    // Insercao na fila de processos prontos

    descritor->prox\_desc = NULL;

    if(prim == NULL){ // -> Fila vazia

        descritor->prox\_desc = descritor;           //Como a fila eh circular, aponta para ele mesmo

        prim = descritor;                           //Ele eh o cabeca de fila

    }

    else{ // -> Fila povoada

        PTR\_PCB aux = prim;                                     //Criacao de um auxiliar para percorrer a fila

        while(aux->prox\_desc != prim) aux = aux->prox\_desc;     //Enquanto a fila nao acabar, percorre a fila

        aux->prox\_desc = descritor;                             //Auxiliar aponta para o processo

        descritor->prox\_desc = prim;                            //Fecha a LCSE apontando para o cabeca de fila

    }

}

// 2. Procurar o proximo processo ativo --------------------------------------------

PTR\_DESC\_PROC Procura\_proc\_ativo(){

    PTR\_DESC\_PROC aux = prim->prox\_desc;               //Criacao de um auxiliar para percorrer a fila de processos

    while(aux != prim){ // -> ate a lista acabar (como eh uma LCSE, isso ocorre quando aux volta a ser prim)

        if(aux->estado == ativo) return aux;           //Achou um processoa ativo -> retorna-o

        aux = aux->prox\_desc;                          //Nao achou -> analisa o proximo processo

    }

    return NULL;                                      //Nenhum processo ativo encontrado

}

// 3. Termina processo -------------------------------------------------------------

*void* far termina\_processo(){

    disable();                                      //Disabilita as interrupcoes

    prim->estado = terminado;                       //Marca processo corrente como terminado

    enable();                                       //Habilita as interrupcoes

    while(1);                                       //Laco eterno

}

// 4. Volta ao DOS -----------------------------------------------------------------

*void* far volta\_DOS(){

    disable();                                      //Desabilita as interrupcoes

    setvect(8, p\_est->int\_anterior);                //Retorna o estado do bit de interrupcao para o padrao

    enable();                                       //Habilita as interrupcoes

    exit(0);                                        //Retorna ao DOS

}

// 5. Escalador de Processos -----------------------------------------------------

*void* far escalador(){

    p\_est->p\_origem  = d\_esc;                       //Origem = escalonador

    p\_est->p\_destino = prim->contexto;              //Destino = prim->contexto

    p\_est->num\_vetor = 8;                           //Bit de interrupcao do timer

    // Inicia ponteiro para regiao critica do DOS

    \_AH=0x34;

    \_AL=0x00;

    geninterrupt(0x21);                             //Gera uma interrupcao

    a.x.bx1 = \_BX;                                  //Bits menos significativos da RC

    a.x.es1 = \_ES;                                  //Bits mais significativos

    // Troca de processos

    while(1){

        iotransfer();                               //Interrupcao de controle

        disable();                                  //Desabilita as interrupcoes

        // Se nao esta na regiao critica, troca os processos

        if(!(\*a.y)){

            if((prim = Procura\_proc\_ativo()) == NULL){ // -> Proximo processo nao existe

                volta\_DOS(); //Volta ao DOS

            }

            // -> Proximo processo existe

            p\_est->p\_destino = prim->contexto;      //Troca os processos

        }

        enable();                                   //Habilita as interrupcoes

    }

}

// 6. Tranferencia do Controle ao Escalador ----------------------------------------

*void* far dispara\_sistema(){

    PTR\_DESC desc\_dispara;

    d\_esc = cria\_desc();

    desc\_dispara = cria\_desc()

    newprocess(scheduler, d\_esc);

    transfer(desc\_dispara, d\_esc);

}

// III - SEMAFORO ================================================================================================

// 1. Definicao semaforo -----------------------------------------------------------

*void* far initiateSemaphore(semaforo \*sem, *int* size\_semaphore){

  sem->s = size\_semaphore; /\* Iniciando o valor com o tamanho dado pelo usuario \*/

  sem->Q = NULL;   /\* Iniciando a fila de bloqueados com nulo \*/

}

// 2. Primitiva P (down) -----------------------------------------------------------

//      -> Decrementa o semaforo. Se o valor chegar em 0, o processo eh bloqueado

*void* far P(semaforo \*sem){

    disable();                              //Desabilita as interrupcoes

    if(sem->s > 0){ // -> regiao critica em uso

        sem->s--;                           //Decrementa S

        enable();

    }

    else{

        PTR\_DESC\_PROC p\_aux;                //Descritor auxiliar

        prim->estado = bloqueado;           //Bloqueia o processo na fila sem->Q

        if(sem->Q == NULL){ // -> Fila de bloqueados vazia

            sem->Q = prim;                  //Insere o processo como cabeca de fila

        }

        else{ // -> Fila povoada

            PTR\_DESC\_PROC aux;              //Auxiliar para percorrer a fila de bloqueados

            aux = sem->Q;

            while(aux->fila\_sem != NULL) aux = aux->fila\_sem;   //Percorre a fila ate seu fim

            aux->fila\_sem = prim;           //Salva o processo na fila

        }

        prim->fila\_sem = NULL;              //Semaforo do processo atual torna-se nulo

        // Procura o proximo processo pronto

        p\_aux = prim;

        if((prim = Procura\_proc\_ativo()) == NULL){  // -> Nao encontrou mais processos ativos

            volta\_DOS();                    //DEADLOCK -> processo atual se bloqueou e nao ha nenhum outro ativo

        }

        // Encontrou -> contexto passa para o novo processo ativo

        transfer(p\_aux->contexto, prim->contexto);

    }

    enable(); //Habilita as interrupcoes

}

// 2. Primitiva V (up) -------------------------------------------------------------

//      -> Incrementa o semaforo. O coloca como ativo se nulo e tiver algum processo na fFila.

*void* far V(semaforo \*sem){

        disable();  //Desabilita as interrupcoes

        if(sem->Q == NULL){ // -> Fila de bloquados nula

            sem->s++;  //Incrementa o semaforo

        }

        else{

            PTR\_DESC\_PROC p\_prox;   //Processo uxiliar

            // Avanca para o proximo da fila de bloqueados

            p\_prox = sem->Q;

            sem->Q = p\_prox->fila\_sem;

            // Remove o processo da fila de bloqueados e o ativa

            p\_prox->fila\_sem = NULL;

            p\_prox->estado = ativo;

        }

        enable();   //Habilita as interrupcoes

}

1. **Estrutura de Dados**
2. **Casos de Teste**