Sistemas Operativos

Licenciatura em Engenharia Informática

1º Trabalho

Simulador de Sistema Operativo

André Gonçalves, 58392 | André Zhan, 58762 | Gonçalo Carvalho, 51817

Departamento de Informática
Universidade de Évora
Abril 2025

Conteúdo

1 Introdução **2**
1.1 Definição dos Estados e Estrutura do Processo 2
1.2 Criação de Processos 2

2 Execução dos Estados **3**
2.1 Estado NEW 3
2.2 Estado READY 3
2.3 Estado RUNNING 3
2.4 Estado BLOCKED 4
2.5 Estado EXIT 4

3 Escalonamento **4**
3.1 Atualização do Estado dos Processos 4

4 Saída do Programa **6**
4.1 Exemplo do output 6

5 Conclusão **6**

1 Introdução

Este relatório descreve a implementação de um simulador de Sistema Operativo em linguagem C, baseado num modelo de cinco estados: NEW, READY, RUNNING, BLOCKED e EXIT. O simulador segue um algoritmo de escalonamento Round Robin com quantum de 3 unidades de tempo e executa um conjunto de instruções com comportamentos distintos como EXEC, JUMP, I/O e HALT. O objetivo é controlar a evolução do estado dos processos ao longo do tempo, respeitando todas as regras definidas no enunciado.

1.1 Definição dos Estados e Estrutura do Processo

```
typedef enum {
    STATE_NEW, STATE_READY, STATE_RUNNING,
    STATE_BLOCKED, STATE_EXIT
} ProcessState;

typedef struct {
    int id, program, pc, quantum, wait_time, block_time, exit_time,
  terminated;
    ProcessState state;
} Process;
```

Cada processo é definido por um identificador único, o índice do programa que executa, um contador de instruções (PC), o estado atual, e variáveis auxiliares para controlo de quantum, bloqueio e terminação.

1.2 Criação de Processos

```
Process* createProcess(int program, int id) {
    Process *p = (Process*)malloc(sizeof(Process));
    p->id = id; p->program = program; p->pc = 0;
    p->quantum = QUANTUM;
    p->wait_time = p->block_time = p->exit_time = 0;
    p->state = STATE_NEW; p->terminated = 0;
    return p;
}
```

A função `createProcess` aloca memória para um novo processo e inicializa todos os seus parâmetros. O estado inicial é sempre `NEW`.

2 Execução dos Estados

2.1 Estado NEW

Processos recém-criados entram na fila NEW e devem permanecer no estado NEW durante dois instantes. Esse comportamento é implementado com o campo `wait_time`:

```
    proc->wait_time++;
    if (proc->wait_time >= 2) {
        enqueue(readyQueue, proc);
    } else {
        enqueue(tempNew, proc);
}
```

Apesar de no segundo instante o processo ser movido para a fila Ready, neste mesmo instante o estado do processo ainda será NEW. Além disso, implementámos o simulador de forma a que apenas no terceiro instante, este mesmo processo, que foi criado no primeiro instante, possa eventualmente ser movido do estado READY para o estado RUNNING, se tiver prioridade na fila.

2.2 Estado READY

Processos em READY aguardam para serem escalonados. Quando o CPU está livre, o processo no topo da fila READY é movido para RUNNING:

```
if (runningProcess == NULL && !isEmpty(readyQueue)) {
    runningProcess = (Process*)dequeue(readyQueue);
    runningProcess->state = STATE_RUNNING;
    runningProcess->quantum = QUANTUM;
}
```

2.3 Estado RUNNING

O processo em execução tem o seu quantum decrementado a cada instante. Cada instrução é analisada:

- **Instrução HALT (0)**: muda o processo para o estado EXIT.
- **Instrução JUMP (101–199)**: altera o contador de instruções (PC) para trás.
- **Instrução EXEC (201–299)**: cria um novo processo (se possível).
- **Instruções negativas (I/O)**: movem o processo para BLOCKED.
- **Outras instruções**: apenas incrementam o PC.

2.4 Estado BLOCKED

Processos com instruções de I/O ficam bloqueados um número de instantes igual ao módulo do identificador da instrução. Ao terminar esse período, são movidos para READY:

```
proc->block_time--;
if (proc->block_time <= 0) {
    enqueue(readyQueue, proc);
} else {
    enqueue(tempBlocked, proc);
}
```

2.5 Estado EXIT

Processos que terminam executam HALT e permanecem em EXIT durante um instante antes de serem marcados como `terminated`:

```
if (processList[i]->state == STATE_EXIT) {
    processList[i]->exit_time--;
    if (processList[i]->exit_time <= 0) {
        processList[i]->terminated = 1;
    }
}
```

3 Escalonamento

O algoritmo de escalonamento implementado é o *Round Robin* com quantum de 3:

```
if (runningProcess != NULL && runningProcess->quantum <= 0) {
    runningProcess->quantum = QUANTUM + 1;
    enqueue(readyQueue, runningProcess);
    runningProcess = NULL;
}
```

Após o quantum ser esgotado, o processo retorna ao final da fila Ready. O QUANTUM + 1 serve como flag, para no próximo instante do ciclo atualizar o estado do processo de RUNNING para READY.

3.1 Atualização do Estado dos Processos

Durante cada instante de tempo, o simulador verifica e atualiza o estado de cada processo com base em condições internas, como o tempo de bloqueio, tempo de saída e valor do quantum. Esse comportamento está refletido no seguinte bloco de código:

```
for (int i = 0; i < MAX_PROCESSES; i++) {
    if (processList[i] != NULL) {
        if (processList[i]->block_time > 0) {
            processList[i]->state = STATE_BLOCKED;
        }

        if (processList[i]->block_time <= 0 &&
            processList[i]->state == STATE_BLOCKED) {
            processList[i]->state = STATE_READY;
        }

        if (processList[i]->quantum > 3) {
            processList[i]->state = STATE_READY;
            processList[i]->quantum = QUANTUM;
        }

        if (processList[i]->exit_time == 1) {
            processList[i]->state = STATE_EXIT;
        }
    }
}
```

Este código garante a coerência dos estados dos processos ao longo da simulação:

- Se o tempo de bloqueio do processo (`block_time`) for maior que 0, o processo é colocado em estado `BLOCKED`.
- Quando o tempo de bloqueio é menor ou igual a 0, o processo é movido automaticamente de `BLOCKED` para `READY`.
- Caso o valor de `quantum` ultrapasse 3 (o valor padrão), significa que o processo já se encontra na fila Ready. Assim, o estado é atualizado para `READY` e o quantum é reinicializado.
- Se o processo estiver prestes a terminar (com `exit_time == 1`), é movido para o estado `EXIT`.

Este passo é essencial para manter a lógica do simulador correta, permitindo que os processos evoluam entre estados com base no seu comportamento individual e no tempo decorrido.

4 Saída do Programa

A função `print_output` regista o estado de cada processo a cada instante, imprimindo num ficheiro "outputXX.out".

```
void print_output(FILE *out, int t) {
    fprintf(out, "%-13d", t);
    for (int i = 0; i < MAX_PROCESSES; i++) {
        if (processList[i] == NULL) {
            fprintf(out, "%-15s", "");
        } else {
            if (processList[i]->terminated) {
                fprintf(out, "%-15s", "___");
            } else {
                fprintf(out, "%-15s", stateToStr(processList[i]->state));
            }
        }
    }
    fprintf(out, "\n");
}
```

4.1 Exemplo do output

```
time inst    proc1    proc2    proc3    proc4
1            NEW
2            NEW
3            RUN        NEW
4            RUN        NEW
5            RUN        READY
6            BLOCKED    RUN
7            BLOCKED    RUN        NEW
...
```

5 Conclusão

O simulador desenvolvido respeita todas as regras especificadas no enunciado, garantindo o controlo dos estados dos processos, o escalonamento por Round Robin, e a criação de novos processos por EXEC.