Teste 10 16/17

1

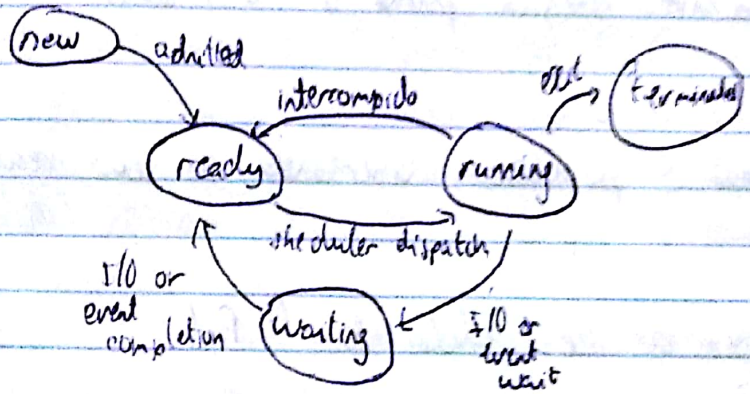
a) F - 4, 5, 10
P - 3, 6, 7, 8, 10

b) A B C ou B A C

c) Tanto os processos como as threads são sequências de execução independentes. A maior diferença entre estes é que as threads (do mesmo processo) correm num espaço de memória partilhado, enquanto os diferentes processos correm em espaços de memória separados.

Assim, percebe-se que as threads têm o melhor desempenho, por haver uma partilha de recursos, são mais responsivas, e favorecem o uso de arquitetura multi-processador.

d)



e) 3 - running
6 - "
7 - waiting
8 - ready -> running
10 - ~~terminated~~ running -> terminated

2

a) int left(int f){ return (f-1)%5; }
int right(int f){ " (f+1)%5; }

b) Ao começar o código, ~~bloqueando todos os~~ está-se a bloquear todos os filósofos, pelo condição de o espera circular.

Scanned by CamScanner

c)

```
void print_avail_resources(void){
    S_mutex.down();
    printf("R1 - %d", info->avail_r1);
    printf("\n R2 - %d", info->avail_r2);
    printf("\n R3 - %d", info->avail_r3);
    S_mutex.up();
}
```

IV

a)

P1	P2	P3	P1	P2	P1	P2	P1	P1
20	40	50	70	90	110	130	140	160

b) $(0 + 20 + 40)/3 = 20$ ms

c) First Come First Serve (FCFS) não inclui nenhuma lógica muito complexa, sendo as funções a realizar postas numa queue e resolvendo por ordem. Round Robin realiza cada processo durante q tempo e quando este terminar passa ao próximo processo.

Como RR é preemptive e não ocorre como em FCFS que, para realizar um processo muito intensivo, tem que ficar a realizar esse processo até acabar, ficando os processos por terminar.

No entanto, RR tem a desvantagem de se:

→ q for muito grande acaba por sofrer das mesmas desvantagens que FCFS.

→ q muito pequeno, o número de trocas é muito elevado e a eficiência do processador diminui

Para escalonar aplicações o FCFS é mais indicado, cálculos muito intensivos, o RR é melhor

Scanned by CamScanner

Val	0	1	2	3	4	5	6	7	8	9	10	11
	G	B	Ø	C	F	A	Ø	Ø	D	E	Ø	H

b) Não, existe o risco de alterar para uma zona já ocupada corrompendo o programa todo

c) ~~Programas maiores podem ser~~
~~Eficiência da utilização da memória~~
~~Segurança~~
Mais memória física está disponível, menos I/O é necessário, tendo mais fácil a mudança entre processos; maiores programas podem ser escritos

Scanned by CamScanner

```
void filosofo (void *arg) {
  while (true) {
    think();
    if (f==0) forks[right(f)].down();
    else forks[left(f)].down();
    eat();
    if (f==0) forks[right(f)].up();
    else forks[left(f)].up();
  }
}
```

2

a) Quando um processo termina, os recursos por ele pedidos adquiridos são libertados, então com

	R1	R2	R3
Recursos disponíveis:	0	2	3

Os recursos disponíveis chegam para o P2, então ele começa

RD: 0 1 2

P2: 2 3 1 | 0 0 0

P2 pode terminar o processo, libertando os seus recursos

RD: 2 4 3

Agora o P2 satisfaz as necessidades do P1.

RD: 0 4 3

P1: 5 1 4 | 0 0 0

P1 pode terminar o processo, libertando os seus recursos

RD: 5 5 7

Há recursos suficientes para P3:

RD: 2 4 7

P3: 5 2 0 | 0 0 0

P3 pode terminar.

3) Não existe deadlock, havendo recursos suficientes do tipo R2 existindo a condição de