Salvaguarda e restauro do contexto do processador

```
void save_context (unsigned int pct_index);
void restore_context (unsigned int pct_index);
```

Reserva e libertação de espaço em memória dinâmica

```
void *malloc (unsigned int size);
void free (void *pnt);
```

Inserção e retirada de nós na FIFO

```
void fifo_in (FIFO *fifo, BINODE *val);
void fifo_out (FIFO *fifo, BINODE **valp);
```

Operações sobre semáforos

```
unsigned int semcreate (void);
void semdestroy (unsigned int sem_index);
void semdown (unsigned int sem_index);
void semup (unsigned int sem_index);
```

Transição de estado dos processos

```
void dispatch (void);
void timerrunout (void);
void sleep (unsigned int sem_index);
void wakeup (unsigned int sem_index);
```

Scheduling do processador (selecciona o próximo processo a que o processador vais ser atribuído)

```
BINODE *sched (void);
```

1. O algoritmo de *scheduling* adoptado não privilegia nenhum tipo de processos. Porquê? Fundamente claramente a sua resposta.

2. Construa a primitiva `timerrunout`.

3. Construa a primitiva `wakeup`.
   Note que, em cada invocação, só um dos processos bloqueados no semáforo é acordado.

4. Se quisesse privilegiar os processos IO-intensivos relativamente aos processos computacionalmente intensivos, como actuaria? Descreva uma solução implementável dentro do contexto do problema. Não se limite a apresentar ideias vagas.