Teste SO – 26/5/17

II

```
ssize_t readln(int fd, char *buf, ssize_t nbytes){
    ssize_t m = 0;
    while ( read(fd, buf+m, 1) && buf[m]!='\n' && m++ < nbytes);
    buf[m]='\0';
    Returm m;
}

int main (int argc, char *argv[]){
    int m=1, atual=0, counter=0;
    char buffer = (char*) malloc(sizeof(char)*1024);
    int fds[argc-1];
    char pipeN[1024];

    for(int i=0; i<argc-1;i++){
        sprintf(pipeN,"pipe%d",i);
        mkfifo(pipeN, 0666);
    }
    for(int i=0; i<argc-1;i++){
        if(!fork()){
            sprintf(pipeN,"pipe%d",i);
            fds[i] = open(pipeN, O_WRONLY);
            dup2(fds[i],1];
            execlp(argv[i+1], argv[i+1], NULL);
        }
    }
for(int i=0; i<argc-1; i++){
    sprintf(pipeN,"pipe%d",i);
    fds[i] = open(pipeN, O_RDONLY);
}
while ( counter < (argc-1)){
    for(int j=0; j<2 && m!=0; j++){
        m = readln( fds[actual], buffer, 1024);
        write(1, buffer, m);
        write(1,"\n",1);
    }
    if(m==0) counter++;
    atual = (atual+1)%(argc-1); }
}
```

ou

```
#define LER 0
#defin ESCREVER 1

int *pid;
int **fds;
```

```
int readlm (int fd, char *buffer, int size) {
    int R, i=0;
    while ((R = read (fd, buffer+i, 1))>0 &&
           i<size) {
        if(buffer[i] == '\n')
            Return i+1;
        i++;
    }
    Return i;
}
```

```
void pagimas (int nProc, char **nameProc) {
    int i, count = 0, lines, aux;
    int fds [nProc][2], control [nProc];
    char buffer [1024];
    for (i=0; i< nProc; i++) {
        pipe (fds[i]);
        if(!fork()) {
            dup2( fds[i][ESCREVER] ,1);
            close (fds[i][LER]);
            close (fds[i][ESCREVER];
            execlp (nameProc[i], nameProc[i], NULL);
            _exit(-1);
        }
        else{
            close (fds[i][ESCREVER]);
            control [i] = 0;
        }
    }
for (int i=0 ; i< nProc; i++)
    wait(0L);
i=0;
while (count < nProc) {
    lines=0;
    while( lines <10 && control[i]!=1) {
        aux = readlm (fds[i][LER], buffer, 1024);
        [redacted];
```

```
            if (aux==0) {
                contad[i]=1;
            }
            write(1, buffer, aux);
            linhas++;
        }
        if (contad[i]==1)
            count++;
        else count=0;
        i=(i+1)%nProc;
    }
}
```

```
int main(int argc, char **argv){
    paginas(argc-1, argv+1);
}
```

SIGCHL → sinal q um processo recebe quando um filho morre.

III

```
short stop=0;
void found(){
    stop=1;
}
int main(int argc, char **argv){
    int n=atoi(argv[1]); int pipepai[n][2]; int pipefilho[n][2];
    int i, pidsex[n], pidsfi[n], turn=0, R;
    char buffer[1024];
    signal(SIGCHLD, found);
    for(i=0; i<N; i++){
        pipe(pipepai[i]);
        pipe(pipefilho[i]);
        if((pidsfi[i]=fork())==0){
            close(pipepai[i][1]);
            dup2(pipepai[i][0],0);
            close(pipefilho[i][0]);
            dup2(pipefilho[i][1],1);
            execlp("filho",NULL);
        }
    else{
        if((pidsex[i]==fork())==0){
            close(pipefilho[i][1]);
            dup2(pipefilho[i][0],0);
            execlp("filho",NULL);
        }
    }
}
```

```
}
while ((R=Readlm(0,buf,1024)>0){
    if(stop){
        for (i=0; i<N; i++){
            Kill(pidsex[i],SIGKILL);
            kill(pidsfi[i],SIGKILL);
        }
        primtlm("O padrão foi encontrado");
        Return 1;
    }
    write(pipepai[turm%size][1],buf,R);
}
primtlm("O padrao ñ foi encontrado");
Return 0;
}
```

① Mecanismos p/ proteção de recursos:

→ Exclusão mútua: forma de evitar q 2 processos tenham simultaneamente acesso a um processo / recurso partilhado, denominado por secção crítica.

→ Controlo de acesso: só determinados utilizadores é que possuem determinadas permissões que lhes permitem aceder a certas posições de memória, executar alguns programas, etc.

→ Autenticação do usuário (?).

② Os deadlocks acontecem quando um processo nunca consegue sair de um estado de espera, uma vez q está à espera de recursos utilizados por outros processos.
Um exemplo bastante comum de deadlock, é a espera circular q consiste quando 1 processo está à espera de um recurso a ser usado por outro processo, que está por sua vez a esta à espera de um recurso. Para evitarmos este caso em concreto de deadlock temos q forçar o grafo de dependências a ser acíclico para podermos ordenar os recursos.

Um exemplo bastante comum de deadlock, é a espera circular q consiste quando 1 processo está à espera de um recurso a ser usado por outro processo, que está por sua vez está à espera de um recurso. Para evitarmos este caso em concreto de deadlock temos q forçar o grafo de dependências a ser acíclico para podermos ordenar os recursos.

(3) Sim, seria boa ideia usar uma estratégia de escalonamento usando ROUND ROBIN. Pois primeiramente o algoritmo de RR é responsável por gerar um time quantum p/ esse processo, e que neste caso em concreto seria o tempo aproximado a escrever 10 linhas do output dos comandos (ligeiramente superior), assim quando um output acabasse de escrever as 10 linhas do seu output, passaria p/ o fim da queue de espera, e assim o proximo processo da queue podia escrever as suas 10 linhas da queue, este processo iria repetir-se até

cada output não possuir mais linhas p/ escrever.