Testes de Sistemas Operativos.
Normas, Bibliotecas de linguagem C e a Biblioteca Posix threads e NASM Assembly etc.

Nos algoritmos de escalonamento.
A prioridade mais alta é o valor 1. Numa situação de empate entre dois processos especifique a maneira de desempatar escolhida

Funções e Estruturas Miscelâneas dos Standard Libraries <stdio.h> <stdlib.h> <string.h> <time.h> <math,h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...); The functions in the printf() family produce output according to a format
FILE *fopen(const char *filename, const char *mode); fopen opens the file named by filename and associates a stream with it. fopen returns a pointer to be used to identify the stream in subsequent operations.
int fgetc(FILE *stream); fgetc() reads the next character from stream and returns it as an unsigned char cast to an int, or EOF on end of file or error.
void * malloc(size_t size); The malloc() function allocates size bytes of memory and returns a pointer to the allocated memory, malloc() returns a NULL pointer if there is an error.
char *ctime(const time_t *timep); **C**onverts the calendar time t into a string of the form "Wed Jun 30 21:49:08 1993\n"
double sqrt(double x); the square root function
int strcmp(const char *s1, const char *s2); int strncmp(const char *s1, const char *s2, size_t n);
Compares one string to another. performs an unsigned comparison of s1 to s2, starting with the first character in each string and continuing with subsequent characters until the corresponding characters differ or until the end of the strings is reached. Return Values: -1 (s1 less than s2) , 0 (s1 the same as s2) 1 (s1 greater than s2)

The **strncmp** function is similar, except it only compares the first (at most) *n* bytes of *s1* and *s2*.

Leitura e Escritura de Baixo Nivel <unistd.h>

int open(char *filename, int access); Opens a file. On success, open returns a nonnegative integer (the file handle or file descriptor). On error, returns -1. Access Examples : O_RDONLY .
int creat(const char *filename, int perms); Creates a file. . On success, open returns a nonnegative integer (the file handle or file descriptor). On error, returns -1. Permission flag examples : S_IRUSR, S_IWUSR S_IXUSR
int read (int handle, void *buf, unsigned len); read attempts to read len bytes from the file associated with handle into the buffer pointed to by buf. On successful completion, read returns an integer indicating the number of bytes placed in the buffer.
int write (int handle, void *buf, unsigned len); write writes a buffer of data to the file or device named by the given file descriptor, handle, write returns the number of bytes written. write returns the number of bytes written.
int close(int fd); closes a file descriptor, so that it no longer refers to any file and may be reused.
int dup(int oldfd); int dup2(int oldfd, int newfd); duplicate a file descriptor. The dup() system call creates a copy of the file descriptor oldfd, using the lowest-numbered unused file descriptor for the new descriptor. The dup2() system call performs the same task as dup(), but instead of using the lowest-numbered unused file descriptor, it uses the file descriptor number specified in newfd. If the file descriptor newfd was previously open, it is silently closed before being reused.

Controlo de Processos <unistd.h> e <wait.h>

pid_t fork(void); create a new process. The new process (child process) is an exact copy of the calling process (parent process) except for some items such as the process ID. Upon successful completion, fork() returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of -1 is returned to the parent process, nochild process is created, and the global variable errno is set to indicate the error.
int execvp(const char *file, char *const argv[]); The exec family of functions replaces the current process image with a new process image. The execvp() function provides an array of pointers to null-terminated strings that represent the argument list available to the new program. The first argument, by convention, should point to the file name associated with the file being executed. The array of pointers must be terminated by a NULL pointer. If any of the exec() functions returns, an error will have occurred and the return value is -1.
pid_t wait(int *status); The *wait*() function shall suspend execution of the calling thread until status information for one of the terminated child processes of the calling process is available. If *wait*() returns because the status of a child process is available, the function shall return a value equal to the process ID of the child process for which *status* is reported.
int pipe(int *fildes); The pipe() function creates an anonymous pipe -- which is an object allowing unidirectional data flow for interprocess communication, and allocates a pair of file descriptors.
int mkfifo(const char *pathname, mode_t mode); make a FIFO special file (a named pipe) with name pathname. Mode specifies the FIFO's permissions. A FIFO special file is similar to a pipe, except that it is created in a different way. Instead of being an anonymous communications channel, a FIFO special file is entered into the filesystem by calling mkfifo(), any process can open it for reading or writing, in the same way as an ordinary file. However, it has to be open at both ends simultaneously before you can proceed to do any input or output operations on it. Opening a FIFO for reading normally blocks until some other process opens the same FIFO for writing, and vice versa. See fifo(7) for nonblocking handling of FIFO special files.

int brk(void *addr); void *sbrk(intptr_t increment);
brk() and sbrk() change the location of the program break, which defines the end of the process's data segment (i.e., the program break is the first location after the end of the uninitialized data segment). Increasing the program break has the effect of allocating memory to the process; decreasing the break deallocates memory.

Linux System Call Table : Deve assumir esta tabela nos exercícios da disciplina

NR	syscall name	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)
0	read	0x00	unsigned int fd	char *buf	size_t count
1	write	0x01	unsigned int fd	const char *buf	size_t count
2	open	0x02	const char *filename	int flags	umode_t mode
3	close	0x03	unsigned int fd	-	-
60	exit	0x3c	int error_code		

Rotinas (Posix) da biblioteca pthreads <pthreads.h> e <semaphore.h>
int pthread_create (pthread_t * thread, pthread_attr_t *attr, void * (*start_routine)(void *), void * arg);
int pthread_join (pthread_t th, void **thread_return);
int pthread_mutex_lock (pthread_mutex_t *mutex);
int pthread_mutex_unlock (pthread_mutex_t *mutex);
int pthread_mutex_init (pthread_mutex_t * mutex,const pthread_mutexattr_t * attr);
int sem_init (sem_t * sem, int pshared, unsigned int value);
int sem_wait (sem_t * sem);
int sem_post (sem_t * sem);

File MetaData <unistd.h> <sys/stat.h> <fcntl-h>
int stat (char *nome, struct stat *buf) : Obtém informação sobre o ficheiro identificado pelo nome.
struct stat {

```
        mode_t     st_mode;        /* file mode (type, permissions) */
        ino_t      st_ino;         /* i-node number */
        short      st_nlink;       /* number of links */
        off_t      st_size;        /* file size in bytes */
        time_t     st_atime;       /* time of the last access */
        time_t     st_mtime;       /* time of the last data modification */
        time_t     st_ctime;       /* time of the last file status change */
        long       st_blksize;     /* optimal I/O block size for filesystem operations*/
        long       st_blocks;      /* actual number of 512 byte blocks allocated */
}
```

int chmod(const char *pathname, mode_t mode); Changes file access mode.

Assembler
CLI/STD: The assembler instruction to turn off/on the interruptions flag, disabling maskarable interruptions.

nasm x86 Assembly Quick Reference

Mnemonic	Purpose	Examples

mov *dest,src*	Move data between registers, load immediate data into registers, move data between registers and memory. dest=destiny src=source	mov eax,4 load constant into eax
		mov ebx,eax Copy eax into ebx
		mov ebx,[123] Copy ebx to memory address 123
		mov eax, [x]
		mov [sum], eax

add dest,src	dest=dest+src	add eax, ebx # Add ebx to eax
sub dest,src	dest=dest-src	sub eax, ebx # Subtract ebx to eax

From: https://www.cs.uaf.edu/2010/fall/cs301/support/x86/index.html

Atomic Operations <stdatomic.h>

_Bool atomic_compare_exchange_weak(volatile A *obj, C* expected, C desired);

Atomically compares the contents of memory pointed to by `obj` with the contents of memory pointed to by `expected`, and if those are bitwise equal, replaces the former with `desired` (performs read-modify-write operation). Otherwise, loads the actual contents of memory pointed to by `obj` into `*expected` (performs load operation). Returns the result of the comparison: true if *obj was equal to *expected, false otherwise. Nota que A e C são normalmente do mesmo tipo !