## Manual for Process and Thread Related Function Calls

**pid_t fork()**

- Used to create a new process, called the child process.
- Both parent and child continue executing from the point of the fork() call.
- Returns the child's PID (a positive number) on parent side and 0 on child side.

**pid_t wait(int *status)**

- Used by a parent process to wait for a child process created by fork() to finish.
- When a parent calls wait(), it blocks until one of its child processes terminates.
- **status:** collects the exit status of the child process. If set NULL, the child's exit status is not collected.
- Returns the pid of terminated child or -1 if on error (*you can skip checking this in your answers*).

**int pthread_create(pthread_t *tid, const pthread_attr_t *attr, void *(*fun)(void *), void *arg)**

- Used to create a new thread within the same process and run a function.
- **tid**: Output variable where thread ID is stored.
- **attr**: Thread attributes (*can be set to NULL for defaults*).
- **fun**: Function to run in the new thread.
- **arg**: Argument passed to the function (*can be NULL if there is no argument*).
- Returns 0 on success or non-zero if on error (*you can skip checking this in your answers*).

**int pthread_join(pthread_t tid, void **retval)**

- Used by one thread to wait for another thread to complete.
- **thread**: The thread ID to wait for.
- **retval**: Pointer to a variable to receive the thread's return value (*can be NULL if unused*).
- Returns 0 on success or non-zero if on error (*you can skip checking this in your answers*).

## CPU Scheduling Algorithm Manual

**First-Come, First-Served (FCFS):** The process that arrives first gets the CPU first. Non-preemptive.
**Shortest Job First (SJF):** The process with the shortest CPU burst runs next. Non-preemptive.
**Shortest Remaining Time First (SRTF):** Always run the process with the smallest remaining burst time. If a new process arrives with a shorter burst than the current one, the CPU preempts and switches to it.
**Priority Scheduling:** The process with the highest priority runs first (lower number = higher priority). It can be preemptive or non-preemptive. For preemptive version, if a new process arrives with a higher priority, it preempts the running process.
**Round Robin (RR):** Each process gets a fixed time slice (quantum) in a circular order. After its quantum expires, a process is preempted and moved to the end of the ready queue.

## Manual for POSIX Shared Memory

**int shm_open(const char *name, int flag, mode_t mode) -** *only needed for named shared memory*

- **name**: Name of the shared memory object.
- **flag**: Flags for accessing the file. (*you should set it to O_CREAT | O_RDWR*).
- **mode:** File permissions (*you should set it to 0666 in you answers*).
- Returns file descriptor on success or -1 on error (*you can skip checking error in your answers*).

**int ftruncate(int fd, off_t length) -** *only needed for named shared memory*

- **fd**: File descriptor (*from shm_open()*).
- **length**: Size of shared memory buffer in *bytes*.
- Returns 0 on success or -1 on error (*you can skip checking this in your answers*).

**void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset) –** *for named and anonymous*

- **addr**: Preferred address (*you should set it to 0 for "let kernel choose"*).
- **length**: Size of shared memory buffer in bytes.
- **prot**: Access permissions. Should set to PROT_READ | PROT_WRITE for both named and anonymous.
- **flags**: Named: set to MAP_SHARED. Anonymous: set to MAP_SHARED | MAP_ANONYMOUS.
- **fd**: Named: set to the file descriptor from shm_open(). Anonymous: set to -1.
- **offset**: Start offset in file (*you should set it to 0*)
- Returns address of shared memory or -1 if on error (*you can skip checking error in your answers*).

## Manual for POSIX Message Queue

**mqd_t mq_open(const char *name, int oflag, mode_t mode, struct mq_attr *attr)**

- **name**: Name of the message queue
- **flag**: Sender: *set to O_CREAT | O_WRONLY*. Receiver: set to O_RDONLY.
- **mode:** File permissions (*you should set it to 0666 in your answers. Only needed for sender*).
- **attr**: Pointer to mq_attr structure specifying queue limits.
- Returns message queue descriptor or -1 on error (*you can skip checking error in your answers*).

**int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio)**

- **mqdes**: The message queue descriptor (*from mq_open()*).
- **msg_ptr**: Pointer to the message buffer to send (*remember: cast your pointer to char**).
- **msg_len**: Length of the message in *bytes*.
- **msg_prio**: Message priority (*you can always set it to 0 in your answers*).
- Returns 0 on success or -1 on error (*you can skip checking this in your answers*).

**ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio);**

- **mqdes**: The message queue descriptor (*from mq_open()*).
- **msg_ptr**: Buffer to store the received message. (*remember: cast your pointer to char**)
- **msg_len**: Size of the buffer in *bytes*.
- **msg_prio**: Optional pointer to store message priority (*you can always set it to NULL in your answers*).
- [illegible]