RTX API

2.1 Memory Management

```
void * request_memory_block();
```

- **returns**: a pointer to the next memory block from the heap, or `NULL` if no memory blocks are available

Retrieves a memory block. If there are no memory blocks remaining, the current process state will be switched to `BLOCKED` and released from the processor.

```
int release_memory_block(void * memory_block);
```

- **memory_block**: a pointer to the memory block to release
- **returns**: `RTX_OK` if successful, otherwise `RTX_ERR`

Restores a memory block to the heap. The memory block becomes available for use if requested. If a process with a higher priority than the current process is blocked, that process will preempt the current process and will be given the released memory block.

2.2 Processor Management

```
int release_processor();
```

- **returns**: `RTX_OK` if successful, otherwise `RTX_ERR`

Releases the current process from the processor. The next queued process is then returned from the scheduler and switched in as the current process.

2.3 Process Priority

```
int set_process_priority(int process_id, int priority);
```

- **process_id**: ID of the process to set
- **priority**: priority in [0,3] to set
- **returns**: `RTX_ERR` if priority or ID are invalid, otherwise `RTX_OK`

Sets the priority of the given process. If the new priority is greater than the priority of the currently running process, the current process will be preempted. The null process and interrupt processes cannot be set.

```
int get_process_priority(int process_id);
```

- **process_id**: ID of the process to get
- **returns**: the priority of the given process

Gets the priority of the given process.

2.4 Interprocess Communication

```
int send_message(int process_id, void * message_envelope);
```

- **process_id**: the process to receive the message
- **message_envelope**: a pointer to a message envelope structure
- **returns**: `RTX_OK` if successful, otherwise `RTX_ERR`

Sends a given message envelope to a specified process. This function preempts if the receiving process has a priority greater than the currently running process. Otherwise, the message is appended to the target process' message queue.

```
void * receive_message(int * sender_id);
```

- **sender_id**: gets set to the process ID of the sender, unless set to `NULL`
- **returns**: a pointer to the received message envelope

A blocking receive message. When called, process execution halts until a message is sent to the process that invoked it.

2.5 Timing Services

```
int delayed_send(int process_id, void * message_envelope, int delay)
```

- **process_id**: the process to receive the message
- **message_envelope**: a pointer to a message envelope structure
- **delay**: the message delay before sending, in milliseconds
- **returns**: `RTX_OK` if successful, otherwise `RTX_ERR`

Identical to `send_message` , except a delay parameter is used to specify a delay in milliseconds before the message is actually dispatched.