Assignment 1

Jeetesh Mangwani, 10314
Harshit Maheshwari, 10290
Vinit Kataria, 10807

POSIX Message Queues

mqd_t mq_open(const char *name, int oflag);

- creates a new message queue or opens an existing queue, returning a message queue descriptor for use in later calls
- returns message queue descriptor of the queue opened if successful
- mqd_t is typically an int
- name is null-terminated string of length less than 256
- oflag} is bitwise OR of O_RDONLY (read only mode), O_WRONLY(write only mode), O_RDWR(read-write mode), O_CREAT(create queue if doesn't exist), O_EXCL(if passed with O_CREAT, create queue exclusively if doesn't exist), O_NONBLOCK(open in a non-blocking mode), where the first three are pairwise mutually exclusive

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

- write a message to a queue
- returns 0 if successful; if fails, returns -1 and sets errno to the code of the error that occurred
- mqdes is the message queue descriptor obtained from mq_open()
- msg_ptr is the pointer to the buffer containing the intended message
- msg_len is the length of the message including NULL terminator
- msg_pio is the message priority
- Doesn't block if the queue was opened with O_NONBLOCK

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

- ssize_t stands for signed size
- if successful, returns the no of bytes excluding the null; if fails, returns -1 and sets errno to the code of the error that occurred
- mqdes is same as in the above function
- msg_ptr is the pointer to the buffer in which to write the message in the front of the queue
- msg_len is the length of the msg_ptr buffer
- if msg_prio is not NULL, the contents of *msg_prio are changed to message priority

int mq_close(mqd_t mqdes);

- closes a message queue that was previously opened
- if successful, returns 0; if fails, returns -1 and sets the errno to code of the error that occurred
- mqdes is the descriptor of the message queue to be closed

int mq_unlink(const char *name);

- removes the message queue named “*name*” and deallocates it
- if successful, returns 0; if fails, returns -1 and sets the errno to the code of the error that occurred
- name is the pointer to the place containing the name of the queue to be unlinked

Our implementation

- src/threads/mqueue.c, src/threads/mqueue.h
  - Find the details in the comments too
  - Contains message queue specific data structures, functions, error codes and flags
  - All sys_mq_*() functions are same as mq_*() functions as described above
  - All the functions are same as POSIX interface except for the following functions:
    - void mqueue_init(void)
      - initializes the global static struct list list_of_message_queues
      - initializes the message priority comparison function
      - called by main() in src/threads/init.c
    - struct mqueue *issue_mqueue(const char* name)
      - allocates memory for & initializes a fresh message queue; returns a pointer to mq_open(), it's caller
    - bool (*prio_less_than)(const struct list_elem *a, const struct list_elem *b, void* aux)
      - serves as a pointer to the message-priority comparison function
      - returns true if the message correponding to a has priority less than that of b
      - aux may be unused
  - The data structures are
    - enum mq_open_mode: enumerates the 6 flags for mq_open; they are powers of 2 so as to retrieve individual flags from the bitwise or of passed flags
    - enum mqueue_error_codes: enumerates the error codes for all the 5 functions described in POSIX section
    - static char error_names[64][128]: conains detailed description for each error code
    - static struct list_of_message_queues: contains all the message queues of the system
    - typedef int mqd_t: serves as identifier for queues
    - static unsigned int no_of_message_queues: contains the number of allocated message queues in memory
    - struct mqueue:
      - mqd_t id: the descriptor of the queue
      - char name[256]: the name of the queue
      - struct list list_of_messages: contains the messages of this queue
      - struct list_elem elem: part of list implementation
      - int no_of_messages: no of messages in the queue
      - struct list read_only_threads, write_only_threads, read_write_threads: list of threads who have openend the queue in read, write and read-write modes
      - struct list send_wait_list, recv_wait_list: contains entries for threads waiting on this queue to send/receive messages due to queue being full/empty
    - struct message
      - char* text: contains a pointer to the buffer containing the message
      - size_t length: message length
      - unsigned int priority: message priority
      - struct list list_elem: part of list implementation, see src/lib/kernel/list.c
    - struct wait_list_entry :serves as member of the struct send_wait_list and recv_wait_list
      - struct thread* thread_ptr: contains the pointer to the thread waiting to send/receive
      - struct list_elem elem: part of list implementation
    - struct tid_holder: serves as member for the struct read_only_threads, write_only_threads, read_write_threads list of the struct mqueue
      - tid_t tid: thread id of the thread associated
      - bool blocking: true if openend in blocking mode
      - struct list_elem elem: part of list implementaion
- src/userprog/syscall.c
  - added code to initialize syscall_map[] and syscall_narg[] arrays in syscall_init() [which is called from main in src/threads/init.c]
  - added code to syscall_handler() so as to transfer control to the function registered for input syscall code
- src/lib/user/syscall.c
  - defined a new macro syscall4(NUMBER, ARG0, ARG1, ARG2, ARG3) on similar lines to those defined already
- src/lib/syscall-nr.h
  - added entries SYS_MQOPEN, SYS_MQSEND, SYS_MQRECV, SYS_MQCLOSE, SYS_MQUNLINK to the enum

Working

Following happens when a message queue function mq_*() is called in a thread/process

- mq_*() are defined in src/lib/user/syscall.c
- mq_w() returns syscallN(SYS_MQW, N arguments....), where N is the number of arguments and W is one of {OPEN, SEND, RECEIVE, CLOSE, UNLINK} and w is one of {open, send, receive, close, unlink}

- syscallN(SYS_MQW, N arguments) is a macro defined in src/lib/usr/syscall.c, where N is 0 to 4 [We have implemented syscall4(NUMBER, ARG0, ARG1, ARG2, ARG3) as mq_send() and mq_receive() require four arguments
- syscallN() raises an interrupt with the code 0x30. The interrupt handler transfers the control to syscall_handler() defined in src/userprog/syscall.c
- syscall_handler() uses the syscall_map[] and syscall_narg[] arrays initialised by syscall_init() [which is called in main() in src/threads/init.c]
- syscall_handler(struct intr_frame *f) transfers the control to the function registered for the number present at (uint64_t *)(&f->vec_no) + 3 [this was told by TA Vikar Kushwah]
- typedef int (*handler)(uint32_t, ...): defines the hander function pointers
- static handler syscall_map[30]: contains the map from code to handler
- static int syscall_narg[30]: contains the no of arguments for function corr to code
- syscall_handler() retrieves the particular function pointer from syscall_map[] array
- In our case, the handlers are sys_mq_*() defined in src/threads/mqueue.c
- in case of any error, sys_mq_*() sets the current_thread()->error to one of the codes defined in mqueue.h [we have added an int error to struct thread in src/threads/threads.h] and returns -1 or 0 as the definition dictates

Test files

- We have made two test files : src/tests/threads/message-passing-block.c and src/tests/threads/message-passing-nonblock.c for testing blocking and non-blocking calls respectively
- Find the details in the comments
- To run, type the following commands in src/threads/build
  - make clean
  - make
  - pintos run message-passing-block
  - pintos run message-passing-nonblock