SE350 Operating Systems Documentation

Gaisano, G., Janecka, L., Mak, R., Schneider, C.

March 30, 2015

1 Introduction

Clarisse

- What is this document about
- What is the purpose of the document?
- What does the document contain?

2 Global Variable Documentation

Everyone

- What does each variable store?
- Why is there a variable to store this?
- What do your global data structures look like?
- What functions use it?

Global Variables

VARIABLE_NAME Stores x for y purpose, used by:

- z

Global Data Structures

DATA_STRUCTURE_NAME Stores x for y purpose, used by:

Structure Name	**Purpose**	**Properties**

Queue	generic queue	• **Element *first**: pointer to first element in the Queue. Is NULL if there are no elements. • **Element *last**: pointer to last element in the Queue. Is NULL if there are no elements.
Element	Generic queue element	• **Element *next**: pointer to next element in queue. NULL if the last element • **void *data**: pointer to element data • **void *block**: pointer to memory block element resides in
PCB	**Process Control Block**. Used to store process identification data and status, such as running state and progress in process	**U32 *mp_sp**: stack pointer of process • **U32 m_pid**: process id • **PROC_STATE_E m_state**: current running state of the process • **int m_priority**: process priority (low value = high priority) • **Queue *mailbox**: pointer to the process' mailbox. Contains all messages passed to process. After initialization, remains unchanged, as only the first and last pointers change.

Block	Represents a chunk of free memory. Is returned to the user on request_memory_block	• **int pid**: the id of the process that currently owns the Block. Is NULL if Block is free. • **Block *next**: pointer to the next Block in the free memory block list. Is NULL if block is in use/allocated to a user process.
msgbuf	Stores the message's data	• **int mtype**: the message type (types defined in CONSTANTS section) • **char mtext[size]**: char array containing the message data. Size is dependent on BLOCK_SIZE
Envelope	The header for a message. Contains necessary information for delivery. Is the structure that is passed around in the message-passing system.	• **int sender_id**: pid of sending process • **int destination_id**: pid of destination process • **int time**: message timestamp • **int delay**: time delay to send message (seconds) • **msgbuf *message**: pointer to message data

- z

3 Kernel API

Clarisse

- All kernel fuctions
- What does each function do?
- What does proper use of this function look like?
- What cleanup is necessary afterwards, if any?

- Does my documentation cover its behaviour in all scenarios?
- Is this described more efficiently through pseudo?

4 Interrupts and their Handlers/Processes

Ginelle

Global Variables: timed_q

Major Design Changes

- Should made a special i-process queue instead of putting in them in the ready queue with other user processes
- Not have i-processes depend on message passing. Instead, have the i-process block itself on finishing with input and be unblocked by the interrupt handler.

Questions

- What interrupts are enabled by your OS? Interrupts:

Interrupt Handlers	**Description**	**Functionality**
Timer	Increments a counter after every clock tick. For each second passed, decrements all messages in the delayed timed queue.	• pushes delayed messages to appropriate destination process mailbox when message delay has passed.
Keyboard	captures keyboard input, composes and sends a message to the UART i-process	• prints debugging hot keys to UART1 output. • sends key press to KCD for command processing.

i-process	Description	Active
UART_iprocess	blah	blah
KCD	blah	blah

- **How does the OS handle those interrupts?** For the Timer Handler, once a message's delay had expired, the interrupt pushed the message to the mailbox of the destination process. All the logic for the timer-related interrupts are in the Timer Handler. For the Keyboard Handler, it sends a message to the UART i-process, who then sends that message to the KCD i-process for command decoding. While waiting

for keyboard input, both i-processes are blocked on received, waiting for messages to arrive. All these processes are given the highest priority to ensure that they interrupt any current processes.

- What do your interrupt processes do?
- Does my documentation cover its behaviour in all scenarios?
- Is this described more efficiently through pseudo?

5 System and User Processes

RayMak

- What system processes are in the OS?
- What is the purpose of each system process?
- Waht does each system process do?
- what services do each of the system processes depend on?
- What system processes does each user processes use?

6 Initialization

Lara

- What steps does your OS take to boot?
- What parameters does your OS have?
- How are these parameters tuned?

7 Testing

Lara

- How did you test your code?
- Did you do unit testing?
- What did your tests do?
- Did you use the debugger?
- Was your testing manual or automated?

8 Major Design Changes

Everyone

- What design decisions ended up being a mistake?
- What were the major stumbling blocks?
- What would you do differently if you started over?
- What design issues did your OS have?

9 Timing Analysis

Everyone

10 Conclusion

Clarisse