User guide - Whilenium

Compiling and booting in Simics

If you have already have a binary-file, just type `./simics bin/Whilenium-1.0.simics` and press "c" in Simics to boot Whilenium.

To compile and boot Whilenium go into the root-directory of Whilenium, and type:

```
make clean
make Whilenium
```

When Simics has launched, type "c" in the prompt to boot Whilenium.

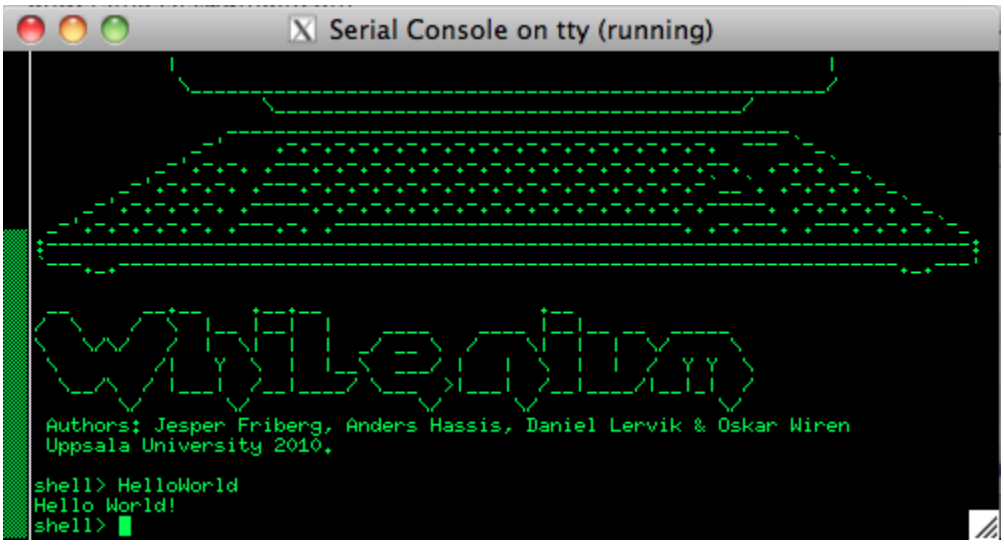
Testing user programs

To launch the different user programs you type their respective names, with capitalized first letter, followed by a " " and the argument for the function (if they take any).

The existing programs are HelloWorld, Increment, Fibonacci, Scroller and Shell. They can be used as follows:

HelloWorld

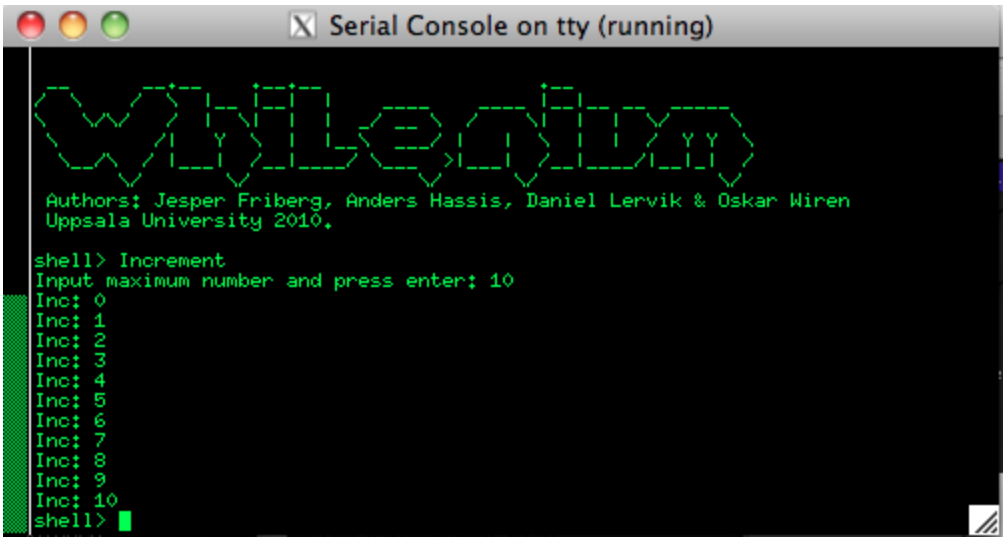
Command: Type "HelloWorld" on the tty display
Description: The text "Hello World" will be printed.



Screenshot of HelloWorld

Increment

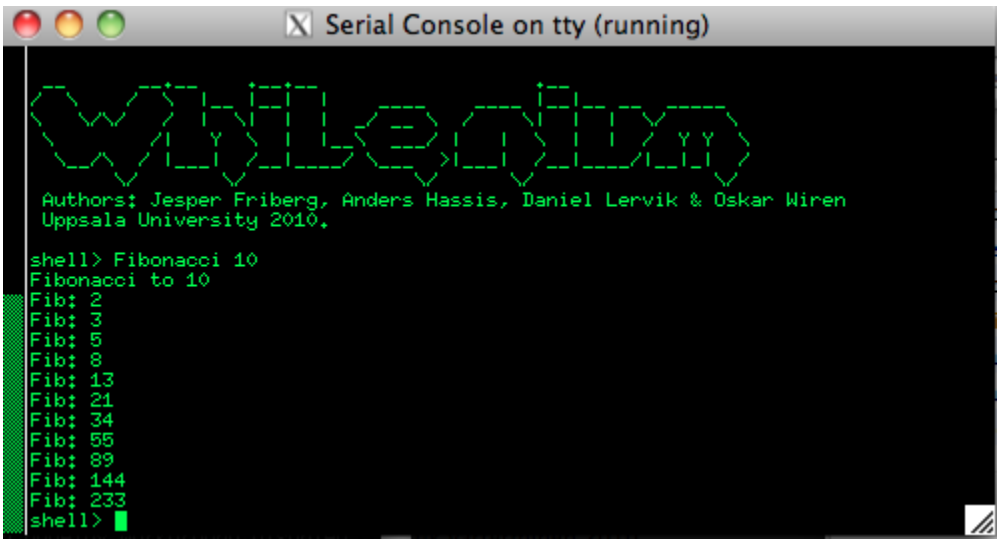
Command: Type "Increment" on the tty display
Description: You will be prompted to enter the number to increment to (n). The program will then increment (0,...,n) and print it



Screenshot of Increment with the number 10

Fibonacci

Command: Type "Fibonacci <n>" on the tty display
Description: Uses the provided argument <n> as input. Prints the n first fibonacci numbers.



Screenshot of Fibonacci with the number 10

Scroller

Command: *Not startable from Shell*
Description: Scroller is started during the bootstrapping process, it isn't

possible to kill this process. You can change the text on the Malta by typing "scroller <text>" where <text> is a text of max 8 chars that should be displayed on the Malta.

Screenshot of Malta display

Shell

Command: Type "Shell" on the tty display
Description: Creates a new shell in which you can execute commands (launch new programs, etc.)
When shell launches a user program the shell is blocked until the new process is terminated. You can create new shell:s in an existing shell.
You can use the "Up-arrow" to retrieve the last command entered, and use backspace to erase the last character on the input.

Miscellaneous commands executable from Shell

top - Prints information about the processes in the system
kill <pid> - Kill process <pid>, if pid = 0 the current process is killed. (pid 1 and pid 2 can't be touched)
block <pid> - Block process <pid>
unblock <pid> - Unblock process <pid>
changePrio <pid> <prio> - Change priority for <pid> to <prio> (pid 1 can't be touched)
scroller <text> - Change the text on the Malta display to <text>
sleep <pid> <time> - Sleep process <pid> for the time <time>

A programmers guide to the Whilenium API

```
/*
 * putc(char c)
```

```
 * Output character c
 * @param char c - Character to output
 * @return 1 if success, -1 if fails
 */
int putc(char c); //

/*
 * puts(char* text)
 * Output string text
 * @param const char* text - String to output
 * @return 1 if success, -1 if fails
 */
int puts(char* text);

/*
 * putsln(char* text)
 * Output text with line-break
 * @param const char* text - String to output
 * @return 1 if success, -1 if fails
 */
int putsln(char* text);

/*
 * gets()
 * Requests a string from input. The process is not running until string is completed with a
'\n'
 * @return char* to the buffer where input is saved for the process
 */
char* gets();

/*
 * displayC(uint8_t word, uint8_t pos)
 * Display a char on the Malta display.
 * @param uint8_t word - Word to display
 * @param uint8_t pos - Position on the display
 * @return 1 if success, -1 if fails
```

```
 */
int displayC(uint8_t c, uint8_t pos);

/*
 * displayNumber(uint32_t word)
 * Display a value on the Malta display.
 * @param uint32_t word - Number to show on the Malta display
 * @return 1 if success, -1 if fails
 */
int displayNumber(uint32_t word);

/*
 * kill(int PID)
 * Kill the process with the given PID with a syscall, process 1 and 2 can't be destroyed
 * If PID = 0, kill the current process
 * @param int PID - Process to kill
 * @return -1 if PID = 1 or 2, or syscall_kill fails. Otherwise 1
 */
int kill(int PID);

/*
 * sleep(int PID, int sleep)
 * Sleep the process PID for sleep iterations, if PID == 0: sleep on current process
 * @param int PID - Process to sleep
 * @param int sleep - Time to sleep
 * @return -1 if fails, 1 if success
 */
int sleep(int PID, int sleep);

/*
 * changePrio(int PID, int prio)
 * Change priority for process PID
 * @param int PID - Process to change
 * @param int prio - New priority
 * @return 1 if succeeded, else -1 for failing
 */
```

```
int changePrio(int PID, int prio);

/*
 * block(int PID)
 * Block process PID
 * @param int PID - Process to block
 * @return 1 if succeeded, else -1 for failing
 */
int block(int PID);

/*
 * unblock(int PID)
 * Unblock process PID
 * @param int PID - Process to unblock
 * @return 1 if succeeded, else -1 for failing
 */
int unblock(int PID);

/*
 * top()
 * Show process information for the whole system
 * @return 1 if succeeded, else -1 for failing
 */
int top();

/*
 * getPrio(int PID)
 * Get priority for process PID
 * @param int PID - Process to get priority for
 * @return 1 if succeeded, else -1 for failing
 */
int getPrio(int PID);

/*
 * getState(int PID)
 * Get state for process PID
```

```
 * @param int PID - Process to get state for
 * @return State Undefined if fails, otherwise it's state
 */
State getState(int PID);

/*
 * getName(int PID)
 * Get name for process PID
 * @param int PID - Process to get name for
 * @return 1 if succeeded, else -1 for failing
 */
char* getName(int PID);

/*
 * spawn(int prio, int PC, char* name, uint32_t arg, State state, int sleep)
 * Spawn a new process with given parameters and run it with a syscall
 * @param int prio - Priority to use, from 0-PRIORITIES
 * @param int PC - Address to the program
 * @param char* name - Name of the program
 * @param uint32_t arg - Argument to pass to our program
 * @param State state - State of the process
 * @param int sleep - If state is Waiting, enter sleeptime here
 * @return Newly created PID on success, else -1 when failing
 */
int spawn(int prio, int PC, char* name, uint32_t arg, State state, int sleep);

/*
 * scroller(char* msg)
 * Display string msg on Malta display
 * @param char* msg - String to display
 * @return 1
 */
int scroller(char* msg);

/*
 * displayS(uint32_t str, uint8_t offset)
```

```
 * Display string S on malta with offset
 * @param uint32_t str - String to display
 * @param uint8_t offset - Offset to use
 * @return 1 if succeeded, else -1 for failing
 */
int displayS(uint32_t str, uint8_t offset);
```