CSE 4300 – Assignment 3

Luke Pepin

Due 3/10/24

Part A:

Files Modified and Changes made:

cs4300-os161/os161-1.11/kern/arch/mips/mips/syscall.c:

Added: SYS\_\_exit case in mips\_syscall and new sys\_exit function.

case SYS\_\_exit:

sys\_\_exit(tf->tf\_a0);

break;

void sys\_\_exit(int exitCode) {

kprintf("Process exited with exit code: %d\n", exitCode);

thread\_exit();

}

Test Program and Output:

cs4300-os161/os161-1.11/testbin/testA/test\_exit.c

#include <unistd.h>

#include <thread.h>

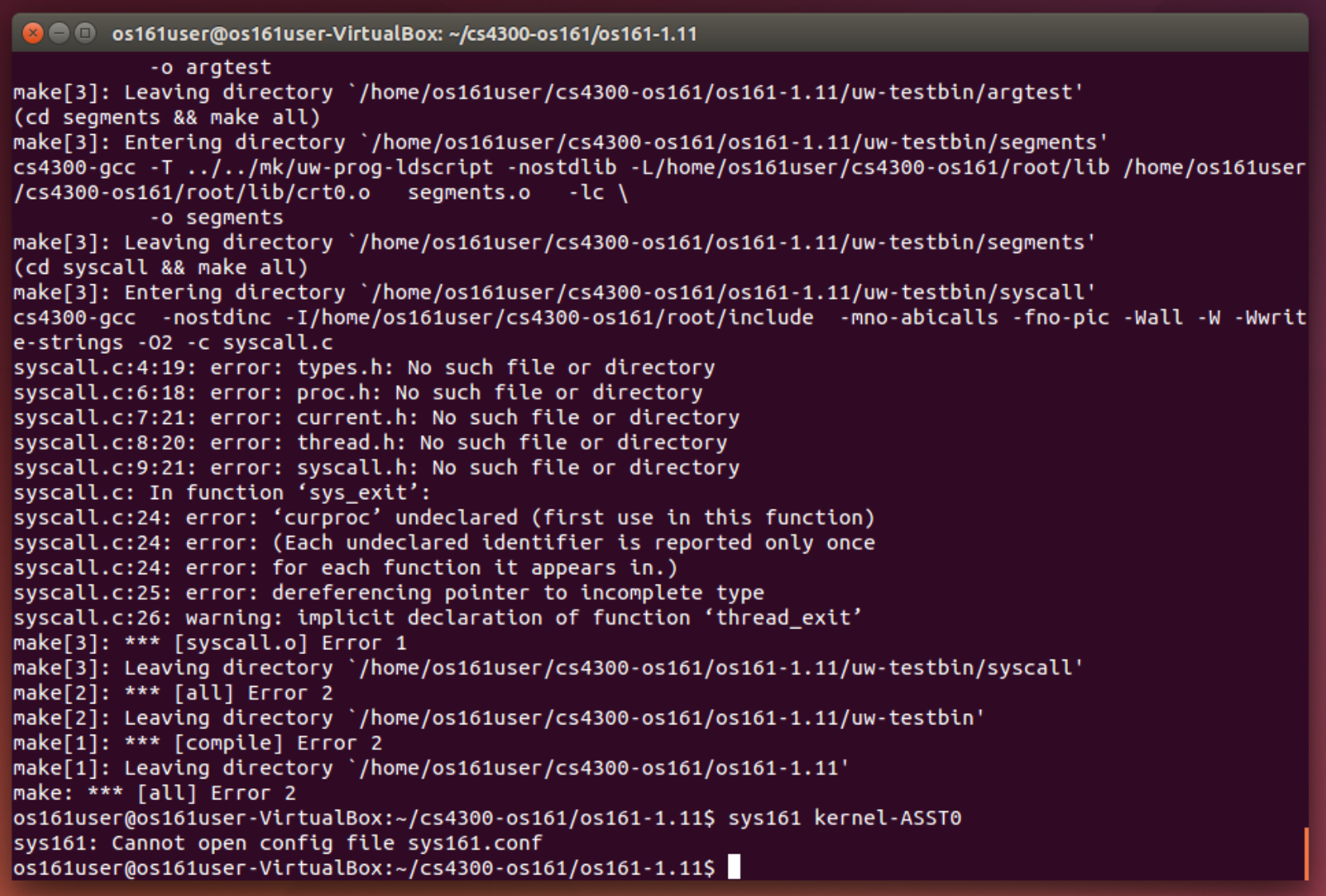
int main() {

\_exit(64);

return 0;

}

After the Makefile was added to testA the following compiling error occured:



Part B:

Files Modified and Changes made:

syscall.c, syscall.h, unistd.h where attempted to add functionality but failed.

Test Program:

#include <stdio.h>

#include <unistd.h>

#include <string.h>

int main() {

int nums[] = {10, 15, 20, 25, 30};

int i;

printf("Testing printint() system call:\n");

for (i = 0; i < sizeof(nums) / sizeof(nums[0]); i++) {

int res = printint(nums[i]);

if (res == 0) {

printf("The number %d is divisible by 5\n", nums[i]);

} else {

printf("The number %d is not divisible by 5\n", nums[i]);

}

}

return 0;

}

Result:

Same Error as Part A

Part C:

Files Modified and Changes made:

syscall.c, syscall.h, unistd.h where attempted to add functionality but failed.

Test Program:

#include <stdio.h>

#include <unistd.h>

#include <string.h>

int main() {

const char \*str = "hello";

int len = strlen(str);

char reversed[len + 1];

printf("\nTesting reversestring() system call:\n");

int rev\_res = reversestring(str, len, reversed);

if (rev\_res == 1) {

printf("Original string: %s\n", str);

printf("Reversed string: %s\n", reversed);

} else {

printf("Failed to reverse string\n");

}

return 0;

}

Result:

Same Error as Part A

Part D:

Files Modified and Changes made:

syscall.c, syscall.h, unistd.h where attempted to add functionality but failed.

cs4300-os161/os161-1.11/testbin/testA/test\_exit.c (for testing part A)

#include <unistd.h>

#include <thread.h>

#include <stdio.h>

#include <stdlib.h>

int main() {

\_exit(64);

return 0;

}

cs4300-os161/os161-1.11/testbin/testA/test\_BC.c (for testing part B and C)

#include <stdio.h>

#include <unistd.h>

#include <string.h>

int main() {

// B

int nums[] = {10, 15, 20, 25, 30};

int i;

printf("Testing printint() system call:\n");

for (i = 0; i < sizeof(nums) / sizeof(nums[0]); i++) {

int res = printint(nums[i]);

if (res == 0) {

printf("The number %d is divisible by 5\n", nums[i]);

} else {

printf("The number %d is not divisible by 5\n", nums[i]);

}

}

// C

const char \*str = "hello";

int len = strlen(str);

char reversed[len + 1];

printf("\nTesting reversestring() system call:\n");

int rev\_res = reversestring(str, len, reversed);

if (rev\_res == 1) {

printf("Original string: %s\n", str);

printf("Reversed string: %s\n", reversed);

} else {

printf("Failed to reverse string\n");

}

return 0;

}

Result:

Same Error as Part A

Part E:

a) How do you enable and disable interrupt in OS161? (4 points)

Through calling splhigh() and spl0(). splhigh() disables interrupts while spl0() enables interrupts.

b) What is the purpose of thread\_yield() function? (4 points)

thread\_yield() allows a thread to give up the CPU. Allowing other threads with equal or higher priority to run on the CPU.

c) What is the difference between thread\_yield() and thread\_sleep() function? (4 points)

Thread\_yield voluntarily gives up the CPU to other threads, but it remains runnable and can be scheduled again. On the other hand, thread\_sleep() removes it from the run queue until it is woken up by another thread or event.

d) What is the purpose of mi\_switch function? (4 points)

The mi\_switch function performs context switches between threads. In other words, it selects the next thread to run from the run queue based on whatever the scheduling policy is and updates the state of the processor to run the thread.

e) Where is the scheduler function located and what does the current scheduler do? (4 points)

It is located in sched.c file in the kernel. It determines the order in which threads are executed on the CPU.