HY345 Assignment 3 Tutorial

System calls for process scheduling parameters

Today's two parts

Part 1 System calls and the lab workflow

Kernel services and how programs request them
Build and boot Linux in the QEMU emulator
Register, call and test a small system call

Part 2 Implementing Assignment 3

Explain the two timing values and their use in scheduling
Store and retrieve them, test both calls and prepare the submission

Part 1

System calls and the lab workflow

Kernel build, QEMU, and a small system call example

Kernel, processes and CPU

- The CPU (Central Processing Unit) executes program instructions.
- The kernel manages CPU time, memory and devices.
- A process is a running program. A Linux task is a unit of execution that the kernel can schedule.
- task_struct is the kernel's record for one task.
- User space contains application code and memory. Kernel space contains privileged kernel code and memory.
- The scheduler chooses a ready task: one that can run and is waiting for the CPU.

System calls, wrappers and the ABI

- A system call lets a user program request a kernel service.
- Arguments are the input values. The kernel checks them and returns a result or an error.
- A wrapper is a small C function that makes the call for a program.
- ABI means Application Binary Interface.
- The ABI specifies the binary rules for passing arguments and return values between the program and kernel.

Examples of kernel services

Calls	What they do
fork, execve	Create a child process / replace its program.
wait, exit	Wait for a child / terminate the calling process.
open, read, write, close	Open a file, transfer data and close it.
getpid, gettimeofday	Read the process identifier / current time.
pipe, socket	Create a channel or endpoint for communication.
chmod, chown	Change file permissions / ownership.

Calling from a C program

libc is the C standard library. Its syscall() function invokes a call by number.
PID means Process Identifier, the number that identifies a process.

```
#define _GNU_SOURCE /* enable libc extensions */
#include <unistd.h>
#include <sys/syscall.h> /* SYS_* syscall numbers */

/* Inside a function: obtain the calling process ID. */
long pid = syscall(SYS_getpid);
```

SYS_getpid selects the service. Other calls may take additional arguments.
For a custom call, its number must match the modified kernel.

Errors: errno and EINVAL

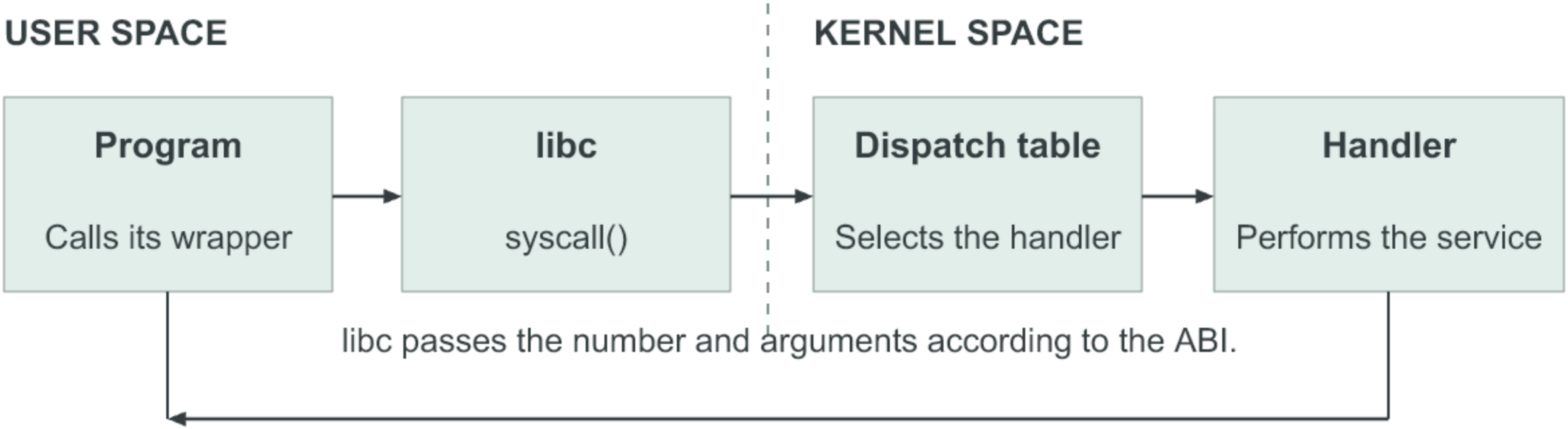
errno is the user-space error indicator. Read it after a failed call.
EINVAL is a named error code meaning “Invalid argument”.

Where	Example of an invalid argument
Kernel code	return -EINVAL;
User program through syscall()	Return value is -1, and errno == EINVAL.

EFAULT: an invalid memory address. ENOSYS: the call is not implemented.
perror("label") prints a readable description of errno.

The system call path

Handler: the kernel function that implements the service.
Dispatch table: maps a syscall number to its handler.



Result returns. On a kernel error, libc returns -1 and sets errno.

The lab host and the QEMU guest

Term	Meaning in this tutorial
Host	The department machine where you run the commands.
QEMU	An emulator: a program that simulates a computer.
Guest	The Linux system running inside QEMU.
Disk image (.img)	A host file that acts as the guest's hard disk.
i386	The 32-bit x86 machine type used by the supplied guest.
root	The administrator account inside a Linux system.

The host and guest have separate filesystems. /home/user is in the guest.

Terms used when building the kernel

Term or file	Meaning
Patched source tree	The kernel source directory with the course’s changes applied.
.config / CONFIG_LOCALVERSION	Build settings / a suffix identifying your kernel version.
GCC / toolchain	GNU Compiler Collection / the compiler and related build tools.
PATH	The list of directories the shell searches for commands.
make / Makefile	The build tool / the file describing what it must build.
Object file (.o) / bzImage	Compiled code to link / the kernel image that QEMU boots.

Linux kernel source

```
# Check this host and its available local space first.
hostname
df -h /spare
mkdir -p /spare/csdXXXX
chmod 700 /spare/csdXXXX
cd /spare/csdXXXX
# Copy and extract the course files.
cp ~hy345/qemu-linux/hy345-linux.img .
cp ~hy345/qemu-linux/linux-2.6.38.1-patched.tar.bz2 .
tar -xjf linux-2.6.38.1-patched.tar.bz2
```

~hy345/qemu-linux/: course archive, disk image, and .config
/spare is local to each host. Use a machine with free space.
Replace csdXXXX with your account; keep this directory private.
Patched tree: the extracted linux-2.6.38.1/ directory.

Kernel configuration and compilation

GCC = GNU Compiler Collection. Use the course GCC 4.9.2 toolchain.

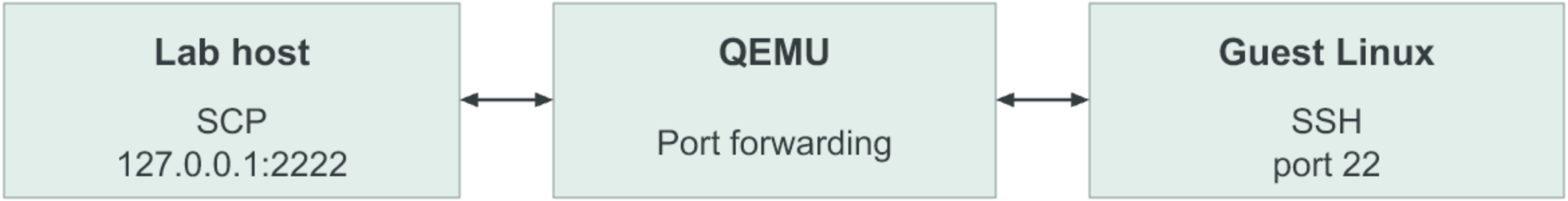
```
cd /spare/csdXXXX/linux-2.6.38.1
cp ~hy345/qemu-linux/.config .
# In .config, set CONFIG_LOCALVERSION, e.g. "-csdXXXX-a3-r1".
TOOLCHAIN=/home/misc/courses/hy345/gcc-4.9.2-standalone
# Select the course compiler and its helper programs.
export PATH="$TOOLCHAIN/bin:$PATH"
export PATH="$TOOLCHAIN/libexec/gcc/x86_64-unknown-linux-gnu/4.9.2:$PATH"
# Check the compiler, then build the 32-bit x86 kernel.
gcc --version
make ARCH=i386 bzImage
```

Build output: arch/x86/boot/bzImage
If the build fails, check the compiler and first error before changing anything.

The host-to-guest connection

SSH (Secure Shell): remote terminal. SCP (Secure Copy): file transfer over SSH.
A port is a numbered endpoint. 127.0.0.1 is the host's own loopback address.

QEMU hostfwd forwards host port 2222 to guest port 22.



Run both SCP commands on the host. Algorithm options allow the old guest's SSH algorithms.
The curses option displays the guest inside the text terminal.

QEMU with the original or modified kernel

Original kernel on the disk image (from /spare/csdXXXX on the host)

```
qemu-system-i386 -hda hy345-linux.img \
  -net nic -net user,hostfwd=tcp:127.0.0.1:2222-:22 \
  -display curses
```

Your rebuilt kernel (from the same host directory)

```
qemu-system-i386 -hda hy345-linux.img \
  -kernel linux-2.6.38.1/arch/x86/boot/bzImage \
  -append "root=/dev/hda" \
  -net nic -net user,hostfwd=tcp:127.0.0.1:2222-:22 \
  -display curses
```

Port 2222 connects the host to the guest. Choose a free port and use it consistently.
Before tests: uname -r must show your CONFIG_LOCALVERSION suffix.
To shut down the guest: /sbin/poweroff (as root).

Files between host and guest

Keep QEMU running with the forwarding in the preceding launch commands.
Run both commands in a second SSH terminal on the same lab host.

```
# Host to guest. Run on the lab machine.
scp -O -P 2222 \
  -o HostKeyAlgorithms=+ssh-rsa \
  -o KexAlgorithms=+diffie-hellman-group14-sha1 \
  test1.c user@127.0.0.1:/home/user/

# Guest to host. Run on the lab machine.
scp -O -P 2222 \
  -o HostKeyAlgorithms=+ssh-rsa \
  -o KexAlgorithms=+diffie-hellman-group14-sha1 \
  user@127.0.0.1:/home/user/test1.c .
```

Guest password: csd-hy345. /home/user is inside QEMU.
Copy the local header and Makefile in the same way; compile inside the guest.

Worked example of a system call

1. Choose an unused i386 system call number.
2. Add its matching dispatch-table entry.
3. Declare the kernel handler.
4. Implement the handler and add its build object.

Rebuild, reboot, and test the new image.

dummy_sys demonstrates the mechanics. It is not an Assignment 3 deliverable.

1 The syscall number

__NR_dummy_sys names the number used to select this call.
NR_syscalls is the table size: the highest syscall number plus one.

arch/x86/include/asm/unistd_32.h

```
#define __NR_dummy_sys 341

/* Update the existing count inside __KERNEL__ */
#define NR_syscalls 342
```

Inspect the supplied tree and choose an unused index.
__NR_ starts with two underscores. __KERNEL__ marks kernel-only declarations.

2 The table entry

Each table entry holds the address of a handler.
.long is an assembly directive that emits one 32-bit entry here.

arch/x86/kernel/syscall_table_32.S

```
.long sys_dummy_sys    /* 341 in this example */
```

Place the entry at the index matching the chosen syscall number.
The handler name must match the C function name.
This guest uses the 32-bit x86 ABI.

3 The handler declaration

A prototype declares a function’s name, return type and argument types.
asmlinkage marks the calling convention expected by the kernel entry code.

include/linux/syscalls.h

```
asmlinkage long sys_dummy_sys(int value);
```

Place it before the final #endif in the header.
Keep the types identical in the declaration and implementation.
The user-space wrapper reaches this handler through syscall().

4 Implementation and kernel messages

printk writes to the kernel log. KERN_INFO marks an informational message.
dmesg displays that log inside the guest.

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/errno.h>

asmlinkage long sys_dummy_sys(int value)
{
    printk(KERN_INFO "dummy_sys value=%d\n", value); /* kernel log */
    if (value < 0 || value > 1000000)
        return -EINVAL; /* invalid argument */
    return 2L * value; /* example: 42 becomes 84 */
}
```

Save as kernel/dummy_sys.c. In kernel/Makefile, obj-y lists built-in objects:
obj-y += dummy_sys.o

A wrapper and a demo program

```
#define _GNU_SOURCE /* expose libc extensions, including syscall */
#include <stdio.h>
#include <unistd.h>
#define HY345_NR_dummy_sys 341 /* must match the kernel table */
static long dummy_sys(int value)
{ return syscall(HY345_NR_dummy_sys, value); }

int main(void)
{
    long result = dummy_sys(42); /* expected result: 84 */
    if (result == -1) { perror("dummy_sys"); return 1; }
    printf("result = %ld\n", result);
    return 0;
}
```

A local header can hold your syscall numbers and wrappers.
perror prints errno when the call returns -1.

Testing the demonstration

```
# Inside the guest after booting the new kernel
uname -r
gcc -Wall -Wextra -o demo.out demo.c
./demo.out
dmesg | tail -n 20
```

Expected positive test: dummy_sys(42) returns 84.
Expected invalid test: dummy_sys(-1) reports EINVAL.
Rebuild and reboot after changing kernel code.

Part 2

Implementing Assignment 3

Required changes, validation, tests, and submission

The two timing parameters

Parameter	Unit	Meaning
deadline	seconds	Allowed duration from a successful declaration.
est_runtime	milliseconds	Estimated CPU execution time needed by the task.

```
deadline = 5 s        est_runtime = 1200 ms = 1.2 s
```

Meaning: the task estimates 1.2 seconds of CPU work within 5 seconds.
Assignment 3 stores these values. Assignment 4 uses them for scheduling.

Deadline duration, D and now

Symbol	Meaning	Example
t0	Clock time of the successful declaration.	10 s
deadline	Declared duration allowed for completion.	5 s
D = t0 + deadline	Absolute deadline: the clock time when time runs out.	15 s
now	The current reading of that same clock.	12 s

D − now = 15 − 12 = 3 s

3 seconds remain before the deadline. Assignment 4 adds the time anchor t0.

Estimated CPU time E and consumed CPU time C

Symbol	Meaning	Example
E	Total estimated CPU time for the declared work.	1200 ms = 1.2 s
C	CPU time already used for that work.	400 ms = 0.4 s
E − C	Estimated CPU time still needed.	1.2 − 0.4 = 0.8 s

Elapsed time: now − t0 = 12 − 10 = 2 s. CPU time used: C = 0.4 s.
Waiting advances the clock, but only running on the CPU increases C.

Assignment 3 stores E as est_runtime. Assignment 4 measures C.

Time before the deadline and work still needed

```
D = 15 s      now = 12 s      E = 1.2 s      C = 0.4 s
```

Quantity	Calculation	Interpretation
D − now	15 − 12 = 3 s	Time available before the deadline.
E − C	1.2 − 0.4 = 0.8 s	CPU execution time the task still needs.

The task needs 0.8 seconds of CPU during the next 3 seconds.
Convert every value to the same unit before comparing or combining them.

Slack: spare time before the deadline

Slack is the available time after allowing for the estimated remaining CPU work.

Slack = (D − now) − (E − C)
= 3 − 0.8 = 2.2 s

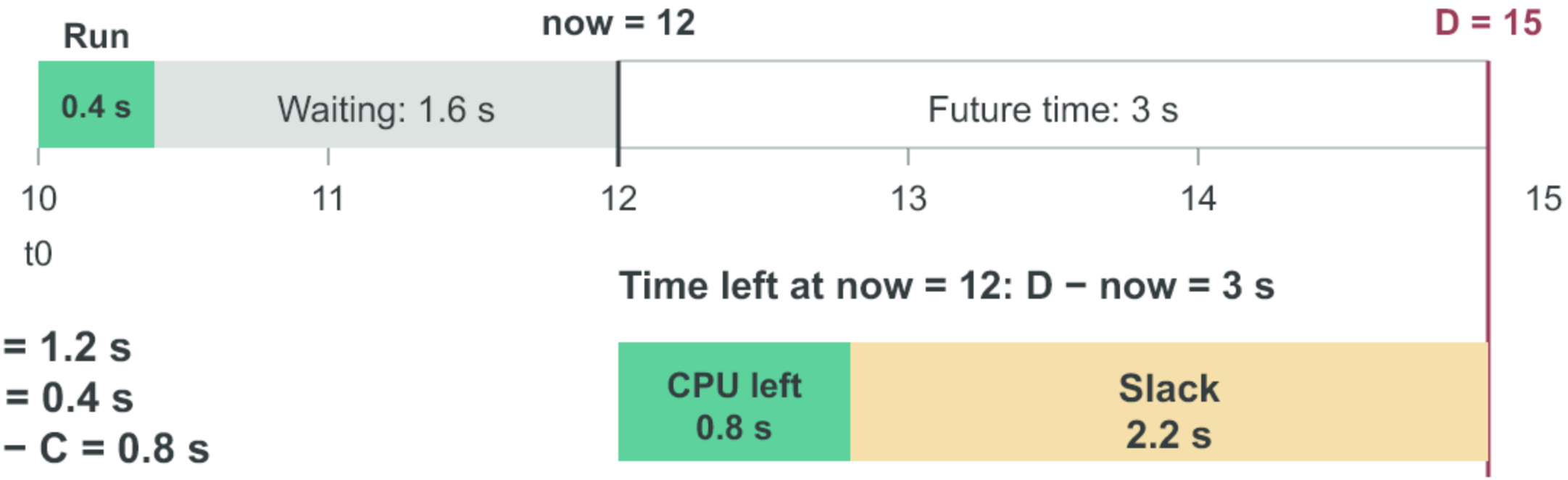
Slack	Meaning under the estimate
Positive	There is time to wait before using the remaining CPU time.
Zero	No spare time. The task needs to run without delay.
Negative	Even continuous execution cannot fit the estimated work in time.

Slack is a calculated quantity. Assignment 3 does not add a slack field.

Timeline: one task and its deadline

t0 = 10 s, deadline = 5 s, D = 15 s, E = 1.2 s

An example history: 0.4 s running, then 1.6 s waiting. We inspect the task at now = 12 s.

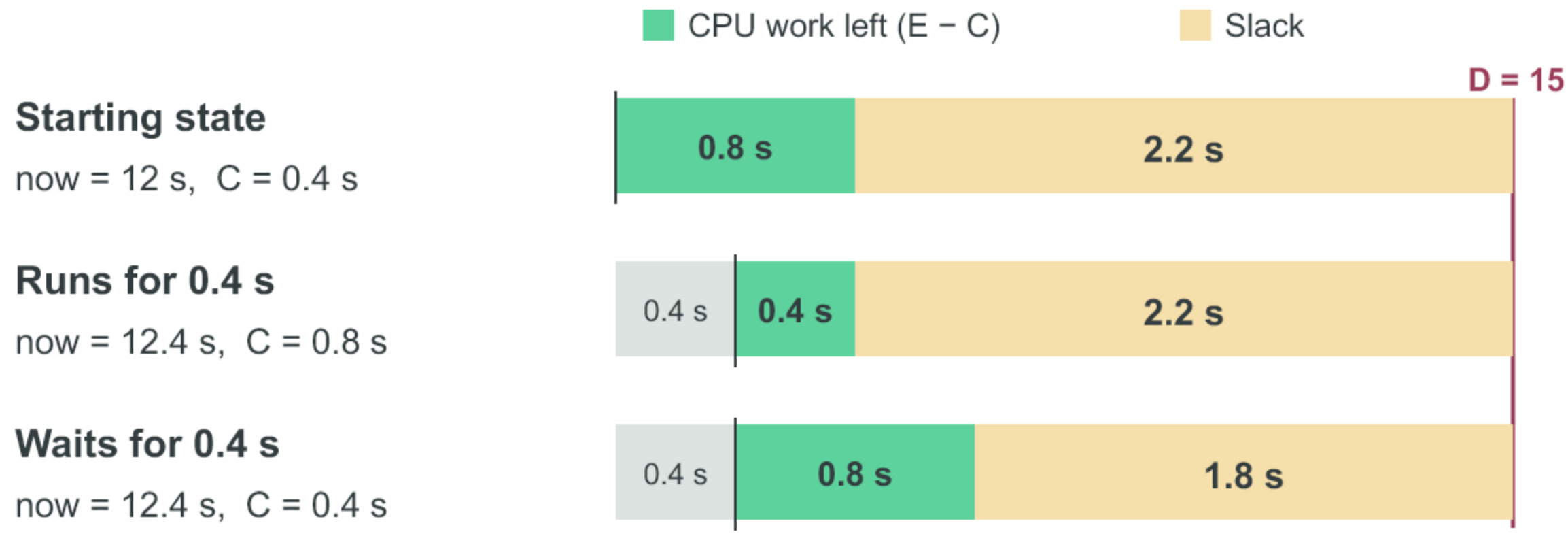


E = 1.2 s
C = 0.4 s
E − C = 0.8 s

The lower bar divides the 3 s available into CPU work and spare time. Slack = 3 − 0.8 = 2.2 s.

Running or waiting for the next 0.4 seconds

Two alternatives from the same state. D = 15 s and E = 1.2 s stay fixed.



Running: time left and work both fall by 0.4 s, so slack stays 2.2 s.
Waiting: work stays 0.8 s, so slack falls to 1.8 s. Grey = the 0.4 s that passed.

Priority in Assignment 4

Priority is a ranking value. The scheduler selects the largest ratio.

Priority = (E − C) / (D − now)

D: absolute deadline. now: current clock reading.
E: estimated CPU time. C: CPU time already consumed. All values below are in seconds.

Task	D	now	E	C	D − now	E − C	Priority
A	14	10	3	1	4	2	0.50
B	16	10	5	1	6	4	0.67

B runs first: 0.67 > 0.50. Evaluate while D > now and E > C.
Expired deadlines and exhausted estimates need Assignment 4 handling.

Assignment 3 requirements

- Add deadline and est_runtime to task_struct.
- Implement set_proc_info to validate and store both values.
- Implement get_proc_info to return them to the calling program.
- Print your name, Student ID and call name with printk in both calls.
- Build on the host; test both calls inside the QEMU guest.

The scheduler and CPU accounting belong to Assignment 4.
The exercise is individual. AI tools and AI-generated code are prohibited.

Store the values in the calling task

```
/* Add to struct task_struct in include/linux/sched.h */
int deadline;     /* relative duration in seconds */
int est_runtime;  /* estimated CPU time in milliseconds */
```

- Access the calling task through current.
- Use per-task fields, not shared global variables.
- An unconfigured task starts with both fields set to 0.
- A child inherits a copy at fork. Later updates remain separate.

The getter’s output structure and user pointer

The getter copies the two stored values into a structure owned by the user program.

```
struct d_params {
    int deadline;    /* seconds */
    int est_runtime; /* milliseconds */
};
```

- The program declares struct d_params params. &params is its address.
- A pointer carries that address. NULL means there is no destination object.
- __user marks a pointer to user-space memory in a kernel declaration.
- Kernel and user code must agree on the fields, their types and their order.

The two system call interfaces

```
/* User-space wrappers */
long set_proc_info(int deadline, int est_runtime);
long get_proc_info(struct d_params *params);

/* Kernel handlers */
asmlinkage long sys_set_proc_info(int deadline, int est_runtime);
asmlinkage long sys_get_proc_info(struct d_params __user *params);
```

- Both calls target the caller; neither takes a process identifier (PID).
- The setter changes both fields. The getter only reads them.
- Success returns 0; a kernel error becomes -1 with errno in user space.

Validate every argument before writing

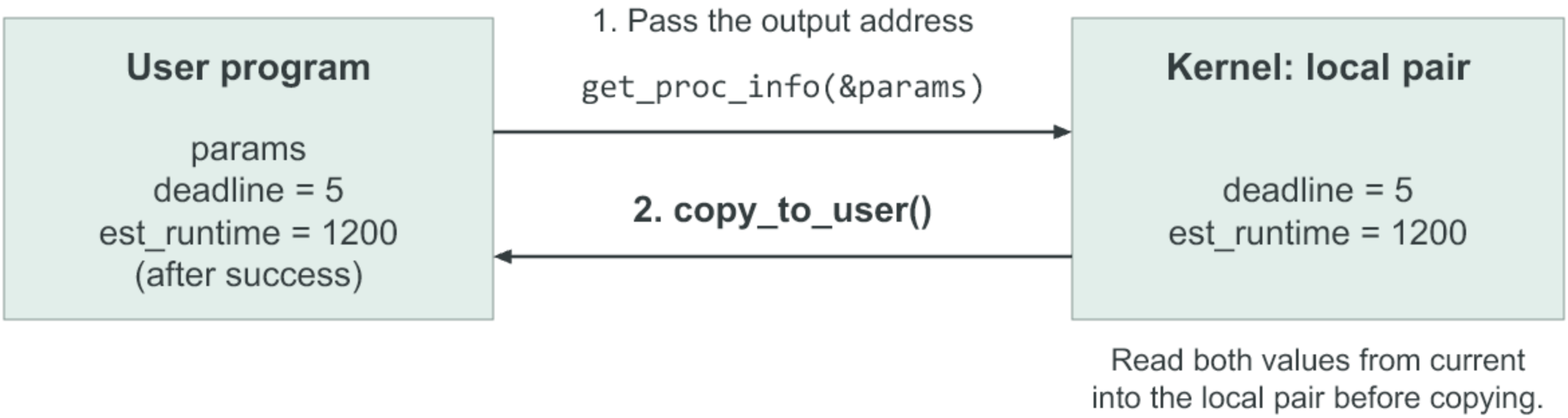
- Require deadline > 0 and est_runtime > 0.
- The declared CPU time must fit within the deadline.
- Equality is valid: (1, 1000) must succeed.

```
est_runtime <= 1000LL * deadline
```

1000LL makes the conversion use long long, avoiding int overflow.
Reject invalid input before assigning either task field.

Safe copying in get_proc_info

copy_to_user() copies bytes from kernel memory to user memory.
It returns the number of bytes that were NOT copied.
The getter leaves task fields unchanged.



NULL: -EINVAL. Incomplete copy: -EFAULT. Complete copy: 0.
Use the output only after success. Never dereference the user pointer directly.

Kernel files to edit or create

Kernel file	Required change
include/linux/sched.h	Add the two task fields.
arch/x86/include/asm/unistd_32.h	Assign two unused numbers; update NR_syscalls.
arch/x86/kernel/syscall_table_32.S	Add both handler entries at their indices.
include/linux/syscalls.h	Declare both handlers and the structure type.
include/linux/proc_info.h	Define struct d_params.
kernel/proc_info.c	Implement both calls (or use separate files).
kernel/Makefile	Add the implementation object or objects.

Match both numbers in the local user header. NR_syscalls = highest number + 1.

Testing the setter and getter together

```
#define _GNU_SOURCE
#include <stdio.h>
#include "proc_info.h" /* wrappers, numbers and struct d_params */

/* Inside main() */
struct d_params params;
if (set_proc_info(5, 1200) == -1) {
    perror("set_proc_info");
} else if (get_proc_info(&params) == -1) {
    perror("get_proc_info");
} else {
    printf("deadline=%d, est_runtime=%d\n",
           params.deadline, params.est_runtime);
}
```

Expected output after both calls succeed: deadline=5, est_runtime=1200.
Use the same structure definition and syscall numbers as the kernel.

Kernel messages and test results

- Both calls print your name, Student ID and system-call name.
- Use get_proc_info to check stored values after successful and rejected setters.
- Check returns and errno on failure. Summarize observations in the README.

```
# Inside the guest
cd /home/user
uname -r
make all
./test1
dmesg | tail -n 30
```

Tests for the two calls

Input or sequence	Expected result
set(5, 1200); set(1, 1000)	Each succeeds; get returns that pair.
Zero or negative setter argument	-1, errno = EINVAL; old pair remains.
set(1, 1001)	-1, errno = EINVAL; old pair remains.
Repeated setters and getters	Each valid setter replaces both fields.
get(NULL)	-1, errno = EINVAL.
get with a non-writable non-NULL address	-1, errno = EFAULT.

One C program may cover all cases. Read stored values with the getter.

Checking that a rejected setter preserves values

Call in the same process	User result	Pair returned by getter
set_proc_info(5, 1200)	0	(5, 1200)
set_proc_info(1, 1001)	-1, errno = EINVAL	(5, 1200)
set_proc_info(8, 2500)	0	(8, 2500)

Call get_proc_info(&params) after each setter and check that it returns 0.
Validate both setter arguments before changing either task field.

Hints and debugging

Symptom	First check
ENOSYS	uname -r, syscall number and i386 table
Positive error code	Return -EINVAL from the kernel
Old behavior after rebuild	Booted bzImage and version suffix
Guest cannot run executable	Compile the test program inside the guest
Values leak between processes	Use fields in current rather than global storage

Kernel modules are code loaded separately from the kernel image.
Module-dependency FAILED after a custom version: see the handout explanation.
With the supplied .config, check that the guest boots and disk/network work.

Submission contents

- The tested bzImage and the .config used to build it.
- All changed or new kernel source files, headers and Makefiles.
- C tests, local headers and a Makefile with all and clean targets.
- A README of at most 30 lines describing the implementation and test observations.

Submit one ZIP on E-Learn and preserve the changed files' relative paths.
Leave out the disk image, full kernel tree and test executables.

Assignment 3 completion checklist

- uname -r identifies the kernel you will submit.
- The setter stores values and the getter returns the expected pair.
- Rejected setters preserve both fields; getter errors are handled safely.
- The ZIP contains the kernel, changed sources, tests, Makefile and README.

Tutorial: 10 November 2026 Deadline: 20 November 2026
Individual work. AI tools and AI-generated code are prohibited.