ECE 8400 / ENGI 9875 Lab 3

Xinyu Jian

TOTAL POINTS

10 / 10

QUESTION 1

1 get_rusage_string() implementation 4 / 4

✓ + 1 pts Source Code

✓ + 3 pts Proof of Implementation: Configuration & Build, and Major & Minor Faults

QUESTION 2

2 Binary execution 2 / 2

✓ + 2 pts Outputs: rusage, after ui_init(), UI thread

QUESTION 3

3 Plots 4 / 4

✓ + 2 pts Proof of Work: Major Page Fault

✓ + 2 pts Proof of work: Minor Page Fault

```
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Looking for pthread_create
-- Looking for pthread_create - not found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /home/xy/ENGI9875/lab3/build
+ ninja
[6/6] Linking C executable lab
+ echo ok
ok
```

4. Build the project by running `ninja` within the `build` directory.

```
$ cd build/
$ ninja -f build.ninja
ninja: no work to do.
```

5. Generate a text file containing 10 MiB worth of pseudo-random data:

```
$ dd if=/dev/urandom bs=1024 count=10240 | base64 > random.txt
```

Command-line `lab` tool

1. Implement the `get_rusage_string()` in `rusage.c`:

```
/**
 * Emit current resource usage information into a string buffer.
 *
 * @param   buffer       a string buffer of at least 1 B
 * @param   len   the length of @b buffer
 */
void get_rusage_string(char *buffer, size_t len)
{
        struct rusage *rptr = malloc(sizeof(struct rusage));
        int rt;
        if((rt = getrusage(RUSAGE_SELF, rptr)) !=0){
                fprintf(stdout, "getrusage() error: %s\n", strerror(errno));
                exit(1);
        }
        snprintf(buffer, len,   "user CPU time used             :
%ld\nsystem CPU time used              : %ld\nmaximum resident set size
: %ld\nintegral shared memory size      : %ld\n"
                                "integral unshared data size    :
%ld\nintegral unshared stack size      : %ld\nminor faults
  : %ld\nmajor faults                      : %ld\n"
                                "swaps                          : %ld\nblock
input operations           : %ld\nblock output operations          :
%ld\nIPC messages sent                 : %ld\n"
                                "IPC messages received          :
%ld\nsingals received          : %ld\nvoluntary context switches        :
%ld\ninvoluntary context switches      : %ld\n",
                                rptr->ru_utime.tv_usec, rptr-
>ru_stime.tv_usec, rptr->ru_maxrss, rptr->ru_ixrss,
                        rptr->ru_idrss, rptr->ru_isrss, rptr->ru_minflt,
rptr->ru_majflt,
                                rptr->ru_nswap, rptr->ru_inblock, rptr-
>ru_oublock, rptr->ru_msgsnd,
                        rptr->ru_msgrcv, rptr->ru_nsignals, rptr->ru_nvcsw,
rptr->ru_nivcsw);

}
```

2. Execute the `lab` binary and record all output.

```
xy@xy-vm ~/D/build> ./lab
----
beginning of main()
----
rusage:
user CPU time used      : 885
system CPU time used        : 0
maximum resident set size   : 2040
integral shared memory size : 0
integral unshared data size : 0
integral unshared stack size    : 0
minor faults            : 90
major faults            : 0
swaps               : 0
block input operations      : 0
block output operations     : 0
IPC messages sent       : 0
IPC messages received       : 0
singals received        : 0
voluntary context switches  : 0
involuntary context switches    : 0

----
after ui_init()
----
rusage:
user CPU time used      : 960
system CPU time used        : 0
maximum resident set size   : 2040
integral shared memory size : 0
integral unshared data size : 0
integral unshared stack size    : 0
```

1 get_rusage_string() implementation 4 / 4

- ✓ + 1 pts Source Code
- ✓ + 3 pts Proof of Implementation: Configuration & Build, and Major & Minor Faults

```
        snprintf(buffer, len,   "user CPU time used             :
%ld\nsystem CPU time used              : %ld\nmaximum resident set size
: %ld\nintegral shared memory size       : %ld\n"
                                "integral unshared data size    :
%ld\nintegral unshared stack size      : %ld\nminor faults
  : %ld\nmajor faults                      : %ld\n"
                                "swaps                          : %ld\nblock
input operations           : %ld\nblock output operations           :
%ld\nIPC messages sent                 : %ld\n"
                                "IPC messages received          :
%ld\nsingals received          : %ld\nvoluntary context switches      :
%ld\ninvoluntary context switches      : %ld\n",
                                rptr->ru_utime.tv_usec, rptr-
>ru_stime.tv_usec, rptr->ru_maxrss, rptr->ru_ixrss,
                        rptr->ru_idrss, rptr->ru_isrss, rptr->ru_minflt,
rptr->ru_majflt,
                                rptr->ru_nswap, rptr->ru_inblock, rptr-
>ru_oublock, rptr->ru_msgsnd,
                        rptr->ru_msgrcv, rptr->ru_nsignals, rptr->ru_nvcsw,
rptr->ru_nivcsw);

}
```

2. Execute the `lab` binary and record all output.

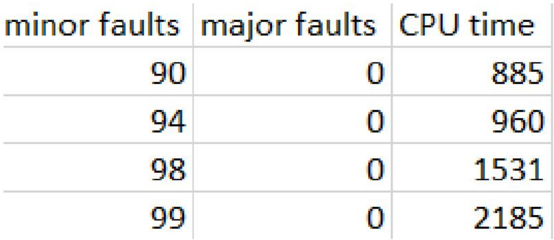
```
xy@xy-vm ~/D/build> ./lab
----
beginning of main()
----
rusage:
user CPU time used      : 885
system CPU time used        : 0
maximum resident set size   : 2040
integral shared memory size : 0
integral unshared data size : 0
integral unshared stack size    : 0
minor faults            : 90
major faults            : 0
swaps               : 0
block input operations      : 0
block output operations     : 0
IPC messages sent       : 0
IPC messages received       : 0
singals received        : 0
voluntary context switches  : 0
involuntary context switches    : 0

----
after ui_init()
----
rusage:
user CPU time used      : 960
system CPU time used        : 0
maximum resident set size   : 2040
integral shared memory size : 0
integral unshared data size : 0
integral unshared stack size    : 0
```

```
minor faults            : 94
major faults            : 0
swaps               : 0
block input operations      : 0
block output operations     : 0
IPC messages sent       : 0
IPC messages received       : 0
singals received        : 0
voluntary context switches  : 0
involuntary context switches    : 0

Press Enter to exit...
----------------------
after starting UI thread
----------------------
rusage information
user CPU time used      : 1531
system CPU time used        : 0
maximum resident set size   : 2040
integral shared memory size : 0
integral unshared data size : 0
integral unshared stack size    : 0
minor faults            : 98
major faults            : 0
swaps               : 0
block input operations      : 0
block output operations     : 0
IPC messages sent       : 0
IPC messages received       : 0
singals received        : 0
voluntary context switches  : 2
involuntary context switches    : 0

----
after UI thread complete
----
rusage:
user CPU time used      : 2185
system CPU time used        : 0
maximum resident set size   : 2040
integral shared memory size : 0
integral unshared data size : 0
integral unshared stack size    : 0
minor faults            : 99
major faults            : 0
swaps               : 0
block input operations      : 0
block output operations     : 0
IPC messages sent       : 0
IPC messages received       : 0
singals received        : 0
voluntary context switches  : 3
involuntary context switches    : 0
```

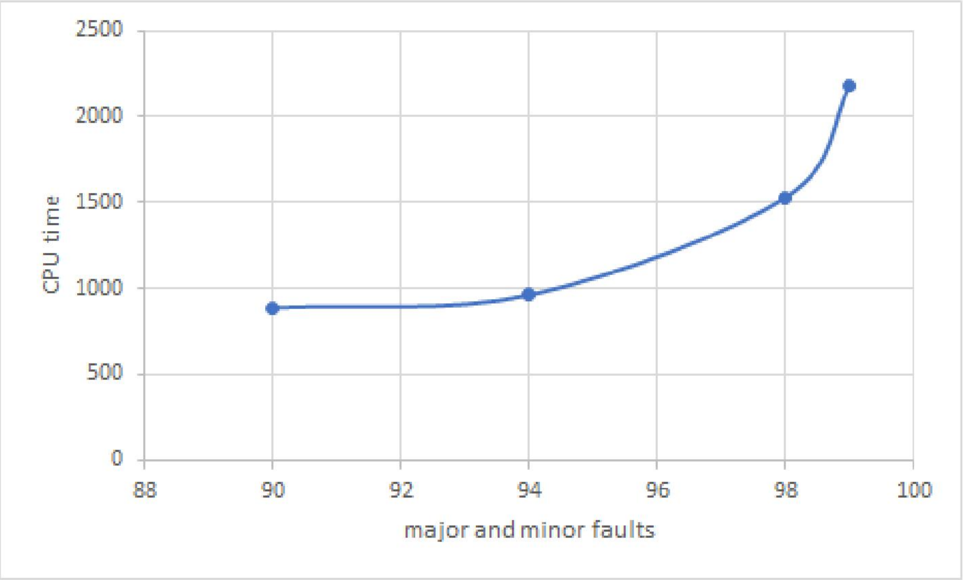
3. Plot page faults — major and minor — against total CPU time.

2 Binary execution 2 / 2

✓ + 2 pts Outputs: rusage, after ui_init(), UI thread



minor faults	major faults	CPU time
90	0	885
94	0	960
98	0	1531
99	0	2185



As we can see, there are no major faults, only minor faults. The reason might be that we run the code just after we created the random.txt, the pages are still in the memory. Thus, there is no I/O activity.

~~GUI~~

1. ~~Execute the `lab` binary again with the `LD_PRELOAD` environment variable set to `gui/libgui.so`. Record all outputs and plot page faults as above. Explain differences with previous tool invocations. What effect does opening (and cancelling) the exit dialog have on rusage?~~
2. ~~Implement `get_command_rusage()` using `wait4(2)` and test it.~~
3. ~~Complete the `on_click_run_*()` functions in `gui/gtk-ui.c` to make them invoke the C compiler (`cc(1)`) and `grep(1)` via `get_command_rusage()`. Recording all outputs and plotting page faults as above, explain the differences (where they exist) between invocations of `cc(1)` and `grep(1)` vs between running `grep(1)` against small and large files.~~

~~Memory pressure~~

1. ~~Use the `top(1)` command to inspect the background memory usage of the computer that you are currently using. How much memory is available? How much swap space?~~
2. ~~Run `/usr/bin/time -v gdb ./lab` to run the lab binary under the `gdb(1)` debugger. Record outputs and plot page faults as above.~~
3. ~~Using the `balloon` application that you completed in your pre-lab preparation, squeeze other data out of memory until the available space is nearly exhausted — in fact, run until the OS won't let you balloon any larger (the machines we'll be working on aren't configured to use swap, so we won't be able to observe the effect of swapping data out). Explain your observations along the way.~~

3 Plots 4 / 4

✓ + 2 pts Proof of Work: Major Page Fault

✓ + 2 pts Proof of work: Minor Page Fault