Fall 2017 CSCI 402

# FAQ for Warmup Project #2

***(Note: this page can change without notice!)***

**Quick index:**

## Commandline

**Q: What's a good way to parse commandline for a real value?**

A: Let's say that the user typed:

```
./warmup2 -lambda 0.25
```

and your code has detected that `argv[1]` contains "`-lambda`". This means that you should parse `argv[2]` for a real value. So, you should make sure that `argc` is at least 3 (which means that `argv[2]` contains a valid string). Then you make sure that `argv[2][0]` is not the "-" character. Then you can "read" a double value from `argv[2]` by doing:

```
double dval=(double)0;
if (sscanf(argv[2], "%lf", &dval) != 1) {
    /* cannot parse argv[2] to get a double value */
} else {
    /* success */
}
```

This is not perfect, but it's good enough!

By the way, for a real value, use **`double`** and never use **`float`**!

## Clock resolution

**Q: Everything in the spec is specified in terms of milliseconds or seconds. How come a timestamp in the output appear to have a resolution of microseconds?**

A: You should report what you **measured**. The actual arrival time of a packet should have microsecond resolution since `gettimeofday()` has a microsecond resolution.

## Basic design

**Q: How do I start this assignment?**

A: Forget about handling <Cntrl+C> for now. I would have the main thread create 4 child threads and immediately join with all of them. When all 4 child threads are dead, the main thread can print statistics and exit the program.

The 4 child threads will collaborate to update the shared data structures (Q1, Q2, token count, and may be some other variables), which is protected by a single mutex. In order to gain access (read or write) to any of these data structures, a thread have to lock the mutex, do something to the data structure, and unlock the mutex. If a thread has to wait for **anything**, you are not allowed to do busy-waiting and must have your thread go to sleep.

You can imagine 4 people trying to accomplish the tasks described by the spec and each person must follow the protocol specified in the spec. What does each person/thread have to do to accomplish the tasks so that things will always work according to the spec? The packet arrival person/thread needs to generate packet in the specified way, then create a packet and add it to Q1 (and in some cases, it needs to then remove this packet from Q1 and move it into Q2). If it move a packet into Q2, it needs to wake up a sleeping server thread. The token depositing person/thread needs to generate tokens in the specified way, then add a token to the token bucket (i.e., just increment the token count), and in some cases, it needs to remove a packet from Q1 and move it into Q2. If it move a packet into Q2, it needs to wake up a sleeping server thread. A server person/thread checks if Q2 is empty or not. If Q2 is empty, it goes to sleep (in a CV queue). If Q2 is not empty, it must remove a packet from Q2 and transmit the packet.

Please remember that whenever you need to use any of the shared data, you must have the mutex locked. What about when the packet arrival thread sleeps to wait for the right time to generate the next packet or when the token depositing thread sleeps to wait for the right time to generate the next token? Well, you should make sure that they go to sleep with the mutex unlocked. What about when a server thread sleeps to simulate the transmission of the packet? Well, you should make sure that it goes to sleep with the mutex unlocked.

## Pthread

**Q: I'm getting -1 (minus one) when I call `pthread_create()`. I think I'm doing everything right. What could be the problem?**

**A:** If you do "`man pthread_create`", you should see:

```
cc -mt [ flag... ] file... -lpthread [ -lrt library... ]
```

This means that you need to do `-lpthread` when you link. On Solaris, you may also want to add `-lnsl` after that.

It is strange that Solaris does not give an error during link time if you do not include "`-lpthread`" but that seems to be the way it is.

---

**Q: What does the skeleton code for the arrival thread look like?**

**A:** **Very roughly**, it looks like the following:

```
for (;;) {
    get inter_arrival_time;
    get service_time;
    usleep(...); /* use inter_arrival_time somehow */
    packet = NewPacket(service_time, ...);
    pthread_mutex_lock(&mutex);
    Q1->enqueue(packet);
    ... /* other stuff */
    pthread_cond_signal/broadcast(&cv);
    pthread_mutex_unlock(&mutex);
}
```

The blue code is inside the **critical section** of the arrival thread.

---

**Q: What does the skeleton code for a server thread look like?**

**A:** **Very roughly**, it looks like the following:

```
    for (;;) {
        pthread_mutex_lock(&mutex);
        while (cur_queue_size == 0 && !shutdown) {
            pthread_cond_wait(&cv, &mutex);
        }
        /* dequeue a job and/or other stuff */
        pthread_mutex_unlock(&mutex);
        /* work on the job based on its service_time */
    }
```

The blue code is inside the **critical section** of the corresponding server thread.

If a thread is executing between the `pthread_mutex_lock()` and `pthread_mutex_unlock()` calls, it is considered to be executing inside a **critical section** with respect to the corresponding mutex.

The call to `pthread_cond_wait()` above is not colored blue for the following reason. If a thread calls `pthread_cond_wait()` inside its critical section, the mutex is released and it is out of the critical section. When this thread receives a signal on the corresponding condition variable, the pthreads library will lock the mutex on its behalf and put it back inside the critical section before `pthread_cond_wait()` returns. So, it should be colored partially blue.

Pthreads **guarantees** that only one thread can be executing in the critical section. So, if the arrival thread is executing inside its critical section, you can deduce that none of the service threads are executing in their critical sections.

## Compiling

**Q: Why am I getting a "too many arguments to function '`sigwait`'" error?**

**A:** If you do "`man sigwait`" on `nunki`, you should see:

```
    SYNOPSIS
         #include

         int sigwait(sigset_t *set);

      Standard conforming
         cc [ flag ... ] file ... -D_POSIX_PTHREAD_SEMANTICS [ library...]
         #include

         int sigwait(const sigset_t *set, int *sig);
```

This tells you that two versions of `sigwait()` are supported. If you want to use the 2nd version of `sigwait()`, you should include "`-D_POSIX_PTHREAD_SEMANTICS`" when you compile your code.

---

**Q: Why am I getting warnings about `flockfile` when I compile?**

**A:** Apparently, `flockfile` and `flockfile` are only available if you insist on POSIX compliance at a certain level. Please include the following when you compile:

```
    -D_POSIX_C_SOURCE=199506L
```

You really don't need to call things like `flockfile()`.

## Running the simulation

**Q: I'm really confused about "inter-arrival time". What does it really mean?**

**A:** "Inter-arrival time" is the time between consecutive arrivals (of packets or tokens). It's a **relative time**. Let's say that if `lambda = 1` (i.e., inter-arrival time for packets is 1 second. For example, when does packet 5 **suppose** to arrives? If you answer "5 seconds after the start of emulation", then you would be wrong. The correct answer is, "I don't know! Tell me when did packet 4 arrived and it's 1 second after that."

---

**Q: If the lambda is 10000, what should be the value of packet interarrival time.**

**A:** It should be 1/10000 seconds = 0.1 millisecond = 100 microseconds.

When your packet arrival thread is ready to sleep, it's possible that you have spent more than 0.1 millisecond to wait for the mutex if the mutex was not available. Then your code should tell you that you need to sleep a negative number of microseconds. So, you should skip sleeping if that is the case. You should not always just skip sleeping if lambda is 10000! You need to do your **measurement** (by reading the clock) and sleep accordingly so your code always works no matter what lambda is.

Please remember that inter-arrival time s a **relative time**. Let's say that if `lambda = 1` (i.e., inter-arrival time for packets is 1 second. For example, when does packet 5 **suppose** to arrives? "5 seconds after the start of emulation" is the wrong answer. The correct answer is, "I don't know! Tell me when did packet 4 **arrived** and it's 1 second after that."

---

**Q: The spec says, "If a server finds Q2 to be empty when it is woken up, it blocks." What does "blocking" mean?**

**A:** The server should wait on a condition variable. A good name for this condition variable could be something like `queue_not_empty`.

If you are not familiar with all these, please read up on waiting and signaling condition variables from Ch 2 of the textbook.

---

**Q: When should the service time of a packet be determined? When the packet was created or when a server picks it up from the queue?**

**A:** Either way would be fine. But it seems to be easier and more uniform if you figure out the service time when a job gets created.

---

**Q: Is using thread-to-thread communication via signals safer or communication via setting a few global variables safer?**

**A:** They are used for different purposes. If a thread is **blocked**, how can it check the value of a global variable?

Well, it cannot. So, you have to wake up the thread. (After it's woken up, it can check the value of the global variable.) The way to wake up a thread is called "signaling it".

But be careful about exactly what it means to "signal" something because there are two types of signals! They are UNIX signals and pthread events (where you can signal an event). You must **read the man pages carefully** and see which type of signal can be used to unblock something.

For example, if you are blocked because you call `pthread_cond_wait()`, your thread will not be woken up if you send a UNIX signal to it. Why? Because if you read the man pages of `pthread_cond_wait()`, it doesn't say anything about UNIX signals!

---

**Q:** **I'm using `printf()` to display state changes as required by the spec. But it looks like the line printed by one thread gets interrupted by another line printed by another thread. How should I fix this?**

**A:** You need a **mutex** for `stdout`. Before you call `printf()`, you should lock the mutex. Unlock the mutex after `printf()` returns.

This seems expensive in terms of extra delays, but it's the price you pay for multithreading.

---

**Q:** **I'm using `usleep()` to sleep for 2000 microseconds. But it looks like it takes 10 milliseconds before `usleep()` actually returns. Is this okay?**

**A:** `nunki.usc.edu` does not run a realtime operating system. So, if you ask to sleep for 2 milliseconds by calling `usleep()`, after 2 milliseconds, your thread will be put on the **ready queue in the OS** for execution. When will your thread actually resume execution? Well, it depends on how many jobs are in front of your thread in the ready queue!

Should it takes 8 extra milliseconds? I don't know because I don't know if there are bugs in your code. You need to determine if this is reasonable! You should write a small bug-free program to see what's a reasonable overhead on nunki.

When you implement your assignment, do **not** anticipate exactly how fast or slow nunki will respond. If `nunki` got an upgrade and became super fast, you must **not** need to change your code in order for your program to run correctly.

---

**Q:** **What's the difference between `usleep()` and `select()`?**

**A:** The `select()` function behaves the same on `nunki` and Linux. On `nunki`, you can do "`man -s 3c select`" to see its man pages.

The man pages of `usleep()` on `nunki` says that it always return 0 when it finishes sleeping. (This is different on Linux!) So, if your thread is blocked on `usleep()` and you send it a signal, it will not return with an error code!

The man pages of `usleep()` on nunki says that, "The `nanosleep(3RT)` function is preferred over this function." The "3RT" means that you need to look for the man pages in the "3RT" section of the man page database. So, do "`man -s 3rt nanosleep`" and you will see that `nanosleep()` can return -1 if there is an error. One of the errors is EINTR which indicates that `nanosleep()` returns because it was interrupted by a signal delivery. So, if you want to use a sleep that can be interrupted by a signal delivery on `nunki`, you should use `nanosleep()`.

---

**Q: My program is not catching <Cntrl+C> every time right after <Cntrl+C> is pressed, what could be the problem?**

**A:** If it's intermittent, most likely, it'e because your code has a race condition.

Let's say that you have something that looks like the skeleton code for the arrival thread. Furthermore, in the SIGINT handler, you set a global variable to inform all your threads that <Cntrl+C> has been pressed and all threads should start the self-termination process. Then you should check this global variable just before and just after the `usleep()` call. But what would happen if <Cntrl+C> is pressed just between checking this global variable and `usleep()`? You will still go into `usleep()`. If you just happen to sleep for a long period of time, your program would behave as if it missed the <Cntrl+C> when the <Cntrl+C> was pressed.

What's the cause of the race condition? It's the same problem as the "Waiting for a Signal" problem mentioned in the slides on "deviations" (section 2.2.5 of textbook). The solution is to first block the signal. Then when the thread is ready, unblock the signal AND wait for the signal using an **atomic operation**!

One solution is to use `pthread_kill()` in the signal handler to send a signal (e.g., SIGUSR1) to the arrival thread (after it has set the global variable). Then in the arrival thread, block SIGUSR1, check global variable, then unblock SIGUSR1 and wait for SIGUSR1 and sleep in one atomic operation with `sigtimedwait()`.

Please note that I do **not** recommand that you use `pthread_kill()`. I also do **not** recommand that you use signal handlers! The best way to go is to use a **SIGINT-catching thread** and use `sigwait()` in it. Make sure you block all signals in all your threads. Please also use the **pthread cancellation** mechanism to terminate the packet arrival and the token depositing threads when <Cntrl+C> is pressed. Please review relevant lectures in Ch 2.

---

**Q: Does a trace specification file has to have an `.txt` file name extension?**

**A:** File name extension has no meaning in Unix. Please do not count on file name extension to mean ANYTHING! `.txt` does **not** mean that it's a text file. It's just part of a file name. It is **incorrect** to reject a file just because it does **not** have a `.txt` file name extension.

---

**Q: Can I assume that all the data in a trace specification file are integers?**

**A:** For a good/valid trace specification file, yes.

---

**Q: How can I know if my code is doing busy-waiting or not? When I ran `top`, it's using less than 1% of the CPU.**

**A:** You cannot tell just by looking at CPU usage. You need to see what your code is doing. If it's not doing anything useful and it's using the CPU to wait for something, then you are doing busy-waiting. If you want to wait for something, you should give up the CPU and wait. So, when your threads are not suppose to be doing anything useful, make sure it's not running (i.e., just sleep for 100 millisecond -- please note that this is the right solution, but it can save you from losing a lot of points).

If your threads are doing something useful, then it's perfectly fine to use the CPU!

---

**Q: When I print the "`begin service`" line, how should I print the amount of service requested? Is it okay to print something like "`requesting 1234.000ms of service`"?**

**A:** Yes. `1234.000ms` and `1234ms` are the same thing.

## Debugging

**Q: When I run my code under `gdb`, <Cntrl+C> is handled by `gdb`. How can I debug <Cntrl+C> handling code under `gdb`?**

**A:** When you start `gdb`, enter the following `gdb` command:

```
handle SIGINT pass
```

---

**Q: How come my code runs perfectly on Ubuntu but crashes on `nunki`?**

**A:** Most likely, it's because you have a memory corruption bug. Please remember that Linux is not the same as Solaris. Their memory allocator is probably completely different and the programs will run differently on these systems. I would suggest that you run your program under `valgrind` on Ubuntu and see what it complains about.

---

**Q: How come my code runs perfectly under `gdb` but crashes without `gdb`?**

**A:** Most likely, it's because you have a memory corruption bug. I would suggest that you run your program under `valgrind` on Ubuntu and see what it complains about.

## Measuring Things

**Q: Do we need to include the bookkeeping time while calculating statistics?**

**A:** You should report what you have **measured**.

---

**Q: Should we report the measured service time and should we used the measured service time to calculate statistics?**

**A:** You should report what you have **measured** and you should calculate statistics with what you have **measured**.

Basically, **everything** after the line emulation begins should all be **measured** values.

---

**Q: How should we calculate the total simulation/emulation time?**

**A:** The total simulation time is the difference between emulation end time and emulation start time.

## Statistics

**Q: What does "average number of packets in a server" mean?**

**A:** When a server is busy serving a packet, the number of packets at the server is one. When a server is idling, the number of packets at the server is zero. In the extreme case, if the server is always busy, the average number of packets at the server should be one. Conversely, if the server is always idle, the average number of packets at the server should be zero. So, in a normal case, the average number of packets should be a real number between zero and one and equal to the fraction of time the server is busy.

---

**Q: What does "average number of packets in Q1" mean?**

**A:** The number of packets at Q1 varies over time, your job is to calculate the average. The trick is to figure out the amount of time Q1 has `x` packets, where `x` can be any integer value. Then you perform a weighted sum of `x` times the fraction of time Q1 has `x` packets, you will get the average.

For example, if the total simulation time is 9 seconds and out of the 9 seconds, Q1 has 0 packet for 2 seconds, 1 packet for 2.8 seconds, 2 packets for 2.5 seconds, and 3 packets for 1.7 seconds. The average number of packets is `(0*2 + 1*2.8 + 2*2.5 + 3*1.7) / 9 = 1.433333`.

So, the trick is for you to keep track of the amount of the time intervals for various queue lengths. But this may end up with too many variables to keep track of. Nevertheless, if you think about it, there is a way to use a constant number of cumulative values instead.

---

**Q: How do you print a real value with 6 significant digits?**

**A:** If you use `printf()`, you can use `"%.6g"`.

If you don't know exactly what "6 significant digits" mean, please google the term "significant digits". For our assignment, it's okay not to show trailing zeroes. For example, 1.2 should be displayed as 1.20000. But it's okay to show it as 1.2 since trailing zeroes is implied. But 0.0000123456 must **not** be displayed as 0.000012 since it would mean that the actual value is 0.0000120000.

---

**Q:** **Do we need to include the dropped packets while calculating the averages?**

**A:** Depends. For example, for average inter-arrival time, you should include them in the calculation. For average time spent in the system, you should exclude them.

---