CIT 5950 Spring 2024: Final

May 10, 2024

First Name : _____Perry_____

Last Name : _____The Paltypus_____

Penn ID : _____

Please fill in your information above, read the following pledge, and sign in the space below:

I neither cheated myself nor helped anyone cheat on this exam. All answers on this exam are my own. Violation of this pledge can result in a failing grade.

Sign Here : _____

Exam Details & Instructions:

- There are 8 questions made of 13 parts (and a short bonus) worth a total of 100 points.
- You have 120 minutes to complete this exam.
- The exam is closed book. This includes textbooks, phones, laptops, wearable devices, other electronics, and any notes outside of what is mentioned below.
- You are allowed two 8.5 x 11 inch sheets of paper (double sided) for notes.
- Any electronic or noise-making devices you do have should be turned off and put away.
- Remove all hats, headphones, and watches.
- **Your explanations should be more than just stating a topic name. Don't just say something like (for example) "because of threads" or just state some facts like "threads are parallel and lightweight processes". State how the topic(s) relate to the exam problem and answer the question being asked.**

Advice:

- Remember that there are 8 questions made up of a total of 13 parts (and a short bonus question). Please budget your time so you can get to every question.
- Do not be alarmed if there seems to be more space than needed for an answer, we try to include a lot of space just in case it is needed.
- Try to relax and take a deep breath. Remember that we also want you to learn from this. A bad grade on this exam is not the end of the world.

Please put your PennID at the top of each page in case the pages become separated.

If you need extra space, the last page of this exam is blank for you as scratch space and to write answers. If you use it, please clearly indicate on that page and under the corresponding question prompt that you are using the extra page to answer that question. Please also write your full name and PennID at the top of the sheet.

PennID: _____

Question 1 {15 pts}

We want to write the function `word_positions()` that takes in a vector of strings as input and returns a map that maps words to what index they show up in the input vector. The resulting map contains each string in the input vector associated with a vector of contains all indexes the string shows up in the input vector.

To be more clear, the following code calls the `word_positions()` function

```
// helper function for printing, assume it works:
string vec_to_str(const vector<size_t>& vec);

int main() {
  vector<string> vec {"luv", "sic", "AND", "NOW", "luv", "sic"};
  map<string, vector<size_t>> res = word_positions(vec);
  for (auto& pair : res) {
    cout << pair.first << ": "l
    cout << vec_to_str(pair.second) << endl;
  }
}
```

When run, it should print out:

```
AND: {2}
NOW: {3}
luv: {0, 4}
sic: {1, 5}
```

It is your job to implement the function word_positions in the box on the next page.

Note that you do not need to worry about the specific ordering of the key values pairs in the map. map<> will automatically sort the entries when they are stored in the map.

Note: You can use any C++ that was covered in class.

You may use this blank space for scratch work. Your answer goes on the next page.

Note: we do not expect people to use all the space provided, but we are providing it just in case it is needed.

```
map<string, vector<size_t>>
word_positions(const vector<string>& input) {
  // this is one possible solution
  // others are possible
  map<string, vector<size_t>> res{};

  for (size_t i = 0; i < vec.size(); i++) {
    res[vec.at(i)].push_back(i);
  }

  return res;









}
```

Question 2 {5 pts}

One of the notable ways we interact with various operating system features is through items we would consider “handles” or “descriptors”. For example, when we open a file, we are given a file descriptor, which is just an integer value. Notably we are not given access to any “real” file data structure directly.

Fork() does a similar thing by returning a process ID instead of direct access to a process control block. A similar thing happens when we create a pthread, the “returned” pthread_t is usually nothing more than an integer that identifies the thread.

Why do you think system calls are designed to give us these identifiers instead of giving direct access to their underlying structures? There is more than one reason, but please give us only one (we will only grade the first reason you list).

You probably don’t need this full space, but we have provided it just in case.

This maintains some separation to make it harder for the user to mess things up or put the OS in an invalid state. e.g. If they could edit the PCB directly then they can put that PCB in an invalid state.

Additional answers:

This maintains a level of abstraction so that the user doesn't need to concern itself with how exactly the operating system manages these things.

If we were to change the PCB structure (or some other internal structure), our code would work because it doesn't deal with it directly, it deals with PIDs and system calls.

Question 3 {15 pts}

One of the most common data structures in computer sciences is a map structure.
Note: you do not need to be familiar with maps, or algorithm analysis to solve this problem. **The maps are just the setting for the question. This question is about memory.**

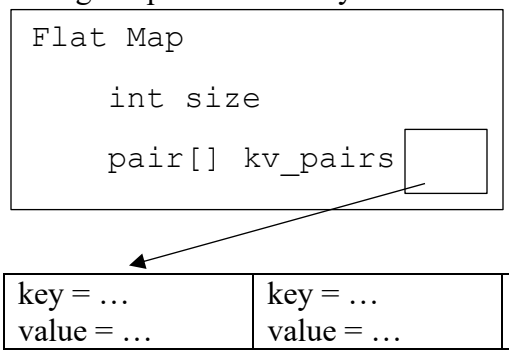
What is a map?

I believe you should be familiar with what a map is from taking the pre-requisite course, but I've included a brief refresher on a map here. Feel free to skip to the memory diagrams if you think you are already familiar. You do not need to be familiar with hashing to answer this question.

A map is a data structure that has two associated types, a key type and a value type. Users can store keys to be associated with a value. A Map is thus a collection of key-value pairs. Common operations include adding a new pairing, setting an existing pairing to have a new value, finding a specific pair from just the key, and iterating over all elements in the map.

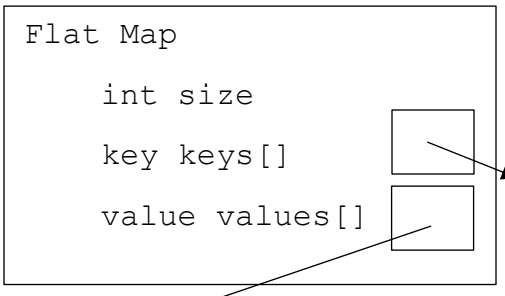
Memory Diagrams

One way we can store key-value pairs is by storing each key with is value as a pair and then storing the pairs in an array. Structures like this are typically called a flat map:



key = … value = …	key = … value = …	key = … value = …	key = … value = …	key = … value = …

A modification we can make on the flat map is that instead of one array, we have two arrays. One array has the keys and another has the values. The key and values are associated by index, so that keys[i] correspond to values[i].



key = …	key = …	key = …	key = …	key = …

value = …	value = …	value = …	value = …	value = …

PennID: _____

Part 1 {10 pts}

Let's say we write code that has a huge map containing many elements. The map uses 4-byte integers as keys and 4-byte floats as values (8 bytes together). We analyze our code and notice that by far the most common operation performed on this structure is the "get" operation:

```
// given a map and a key, looks through the map for the
// presences of the key and returns the corresponding value
value get(map m, key k);
```

If we wanted to maximize performance for that operation, which structure would be better? Why? **Your answer should be 2-3 sentences.**

Hint: if you are thinking about algorithm analysis like O(n) stuff or counting the number instructions executed, you are doing it wrong

Two possible accepted answers:
The flat map with two arrays is faster for this operation due to the fact we only need to check the keys for the find operation and we can fit more keys per cache-line in the two-array structure

1 array would be faster since both the key and value would be in the same cache-line and benefit from spatial locality when we want to return the found value

Part 2 {5 pts}

If the keys and value pairs were larger (let's say that the keys are 64 bit integers (8 bytes) and that the value is the size of a page, 4096 bytes). How does this change your answer from part 1? Please explain why. Limit your answers to 3 sentences at maximum. Note: We will try and treat your answer to part 1 as "correct" for the sake of this question.

The second structure (two arrays) would be better due to the fact we only need to check the keys for the find operation and we can fit more keys per page in the two-array structure. The 1 array structure would result in page fault on every kv pair.

PennID: _____

Question 4 {15 pts}

Travis tries to implement his own mutex using only booleans between two threads.

His attempt is shown below:

```
bool flag1 {false};
bool flag2 {false};
vector<string> global_vec{};

void* first_thread(void* arg) {
  flag1 = true;
  while (flag2) {
    sleep(1);
  }

  // only one thread should execute this function at a time
  critical_code(global_vec);

  flag1 = false;
}

void* second_thread(void* arg) {
  flag2 = true;
  while (flag1) {
    sleep(1);
  }

  // only one thread should execute this function at a time
  critical_code(global_vec);

  flag2 = false;
}
```

You can assume that exactly one thread is created to run first_thread(), exactly one thread is create to run second_thread(), that both threads are created successfully, and that this compiles successfully.

Question continues onto the next page

Part 1 {8 pts}

This code does not work properly every time we run it. This is for a few reasons, but what is one reason it does not always work? Please justify your answer

Note: I don't mean that this is inefficient, I mean that it does not work.

if both flags are set to true before either thread enters a while loop, the program enters deadlock as each thread will wait for the other to set their flag.

Part 2 {7 pts}

Above we said that the code did not work in some cases, but we did not promise it would fail EVERY time we ran it. Is it possible that we run the code, and it runs without issue? Please briefly justify your answer.

It is possible. One such way is if the code is executed such that thread 2 does not start until thread 1 finishes completely.

Question 5 {13 pts}

Ash is writing multi-threaded code and has repeatedly forgotten to unlock a lock when they are done with it. Ash remembers how this is similar to an issue with C, where it's easy to forget to free things, but in C++ it is possible to use a smart pointer that will automatically clean up memory for us when the smart pointer object is destructed.

Ash takes inspiration from smart pointers, and supposes we could implement a new object called a "lock guard" that will automatically acquire a lock and then later unlock the lock when they are done with it.

Part 1 {6 pts}

Ash has started writing the implementation, but it is not yet finished, please finish the implementation:

(Note there are two boxes for you to write code in, please put something in both)

```
class lock_guard {
 public:
  // constructor
  lock_guard(pthread_mutex_t* lock) : lock_(lock) {
      // todo: implement the rest

      pthread_mutex_lock(lock);



  }

  // destructor
  ~lock_guard() {
      // todo: implement the rest

      pthread_mutex_unlock(lock);



  }
 private:
  pthread_mutex_t* lock_;
};
```

Part 2 {7 pts}

Travis tries to use what Ash wrote to write thread safe code that increments a shared global variable. He writes the following code and finds that there is still a data race in this code:

```
int global_counter = 0;
pthread_mutex_t global_mutex;

void* thread_code(void* arg) {
  for (int i = 0; i < 5; i++) {
    lock_guard(&global_mutex);
    global_counter += 1;
  }
  global_counter += 5950;
  return nullptr;
}
```

Please explain what mistake Travis made that lead to a data race in this code?
You can assume the code compiles, that there is more than one thread running thread_code() and that all threads (and locks) are created successfully.

The last adding to the global counter outside of the loop is not protected. This is because the lock guard releases the mutex when it leaves scope, and so when we leave the for loop (and on each iteration), the lock is released and so global_counter += 5950 is not protected.

Question 6 {16 pts}

Part 1 {10 pts}

Suppose we're working on a program that involves sending requests over the network and waiting for the responses the server sends back. After some testing and user feedback, we're told is "too slow". Specifically, there are a lot of things the application needs to load and it's taking a while to do. Because of this, Sean comes up with an idea for how to make our application faster.

Sean supposes we can store recent requests and their corresponding response in the program's memory to avoid having to make the same request again over the network. Is this something that can be implemented, and if it was implemented, would it likely improve performance? Please explain your answer.

This can be implementable with some sort of map and it would make it faster because accessing memory is generally a lot faster than accessing sometihng over the network

Part 2 {6 pts}

For any given connection over the network, can we ensure that the bytes we send will be successfully received at its intended destination and received in the proper order? Please briefly justify your answer.

We can't guarantee delivery/reliability since there might be network partition, or we can simply just unplug the server.

Question 7 {6 pts}

Let's say you've written a program that runs really well and does everything you need to, except that once every day it crashes. Fortunately for you, it's not doing anything critical - but it's not worth the development time to find and fix the cause of the crash.

You decide to write a program that checks the status of another program and restarts it if it crashes. You are deciding whether your two programs (the one that crashes and the one that restarts) should be two threads in the same process or in two separate processes.

Which do you choose? Briefly explain your answer

You need two separate processes because otherwise the two threads share a memory space and if one crashes they both will crash. If we have two processes there is some isolation and thus the program that "restarts" the failing program can keep running when the failing program crashes.

Question 8 {14 pts}

Part 1 {6 pts}

Consider a new page replacement policy called MRU (Most Recently Used). MRU can be thought of as the "opposite" of LRU. Where the page that was most recently used is evicted from physical memory if another page needs to be brought in. (e.g., if page A was most recently accessed, and we then wanted to load in a page not in the physical memory, page A would be evicted to make space for the new page)

Assume we have virtual pages A B C D, start with empty physical memory and physical memory can only hold 2 physical pages.

What is a sequence of page accesses that would result in MRU having less page faults than LRU? **Please give a sequence of accesses that is exactly 6 page accesses long in the boxes below. Other solutions are possible**

Access #	0	1	2	3	4	5
Page accessed	A	B	C	A	B	D

Part 2 {8 pts}

For this problem, lets assume we are working with a 32-bit system. That means the pointer in our virtual address space for a process is 32 bits, and thus there are 2^32 different addresses in our program's virtual address space. If each address corresponds to one byte in memory, and a virtual page is 2^12 bytes, then what do we know about the following items?

Please briefly justify answer for each of these. If we do not have enough information to determine an answer, please explain why we cannot determine that answer.

Item	Amount & short justification
Number of virtual pages per process	**2^{20}** **2^{32} bytes with one address per byte and a page is 2^{12} bytes, 2^{32} bytes / 2^{12} bytes per page is 2^{20}**
Bits needed for the virtual page number	**Since there are 220 virtual pages, we need 20 bits to represent 220 possible different page numbers**
Size of a physical page	**2^{12} since the size of a physical page is the same as the size of a virtual page**
Number of physical pages in physical memory	**We do not know since we do not know the size of physical memory.**

PennID: _____

Question 9 {1 pt} <u>all submissions will get this point</u>

What's an interesting/niche hobby you have? Or is there anything interesting you do that helps destress during the semester?

If you don't want to do that, then put anything here! What's your favourite thing you learned this semester? Anything you want to show us or want us to know?

Music helps me destress. I would recommend this album for destressing:

Appendix

pthread_create

SYNOPSIS

```
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
```

DESCRIPTION

The pthread_create() function starts a new thread in the calling process. The new thread starts execution by invoking start_routine(); arg is passed as the sole argument of start_routine().

pthread_join

SYNOPSIS

```
int pthread_join(pthread_t thread, void **retval);
```

DESCRIPTION

The pthread_join() function waits for the thread specified by thread to terminate. If that thread has already terminated, then pthread_join() returns immediately.

If retval is not NULL, then pthread_join() copies the return value of the target thread into the location pointed to by retval.

pthread_mutex_lock

SYNOPSIS

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

DESCRIPTION

The mutex object referenced by mutex shall be locked by calling pthread_mutex_lock(). If the mutex is already locked, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by mutex in the locked state with the calling thread as its owner.

pthread_mutex_unlock

```
SYNOPSIS

          int pthread_mutex_unlock(pthread_mutex_t *mutex);

DESCRIPTION

The pthread_mutex_unlock() function shall release the mutex
object referenced by mutex.
```

iterator find(iterator begin, iterator end, T target);

```
Given a range of values specified by the begin and end
iterators, searches with the range starting at begin and ending
at (but not including) end for the specified target. If the
target value is found, then it returns an iterator to that
element. If it is not found, then end is returned.
```

size_t vector<T>::size();

```
Member function for the vector class. Returns the number of
elements in the vector.
```

void vector<T>::push_back(const T& value);

```
Member function for the vector class. Called on a vector to push
the specified value onto the end of the vector, thus extending
it to be 1 element larger.
```

T& vector<T>::at(size_t index);

```
Member of the vector class to access an element at the specified
index of the vector.
```

V& map<K, V>::operator[](const K& key);

```
Member of the map class to access the value associated with the
specified key. If the key-value pair does not exist already in
the map, then it is implicitly inserted using the default value
for the value and returning a reference to the value.
```

bool map<K, V>::contains(const K& key);

```
Member of the map class to see if the specified key exists in
the map. Returns true if it does, false if it does not.
```

This page is intentionally left blank.