CSIE 3310, Spring 2021 — Midterm Solution Sketch

Name						SID									

- Do NOT start when you get the exam sheets — Wait!
- You should write your answers in the specific areas.
- You should provide sufficient explanation, except "no explanation is required" is mentioned.
- You should show your intermediate steps for the calculation questions.
- Turn in the exam sheets and your smart phone to the TA(s) if you want to go to the restroom.
- Do NOT go to the restroom in the last 10 minutes.
- Do NOT turn in your exam sheets in the last 10 minutes.

1 Term Definitions (20pts)

1. (2pts) Multitasking.

   Answer: Multiple processes are executed on the same processors and are switched frequently to emulate the single tasking environment.

2. (2pts) Bootstrap Program.

   Answer: A program to load the operating system into the memory and processor when the power is turned on.

3. (2pts) Non-Volatile memory.

   Answer: The memory devices can keep the content without power.

4. (2pts) Middleware.

   Answer: A set of software frameworks that provide addition services to application developers such as databases, multimedia, graphics.

5. (2pts) Process Control Block (PCB).

   Answer: The data structure in operating system to store information associated with each process.

6. (2pts) von Neumann Architecture.

   Answer: A stored-program computer which fetches instructions and data from the storage (memory), executes the instructions on the processors, and generate outputs to the device.

7. (2pts) Internal Fragmentation.

   Answer: The size difference between the memory allocated to a process and the requested memory.

8. (2pts) Copy-on-Write.

   Answer: The copy-on-write technique allows the parent and child processes initially to share the same pages, and, if either process writes to a shared page, a copy of the shared page is created.

9. (2pts) Belady's Anomaly.

   Answer: For some page-replacement algorithms, the page-fault rate may increase as the number of allocated frames increases.

10. (2pts) Thrashing.

    Answer: A process is thrashing if it is spending more time paging than executing.

2 Short Answers (40pts)

1. (4pts) Compare the difference(s) between *I/O Bound Process* and *CPU Bound Process*.

   Answer: An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations. A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations.

2. (4pts) Compare the difference(s) between *Implicit Threading* and *Explicit Threading*.

   Answer: Implicit threading is a thread programming environment where the programmers only design function logics but *do not manage the threads*, including thread creation, thread scheduling, and result assembling. Explicit threading is a thread programming environment where the programmers design function logics as well as *manage threads*.

3. (4pts) Compare the difference(s) between caching on *Multi-Core Processor* and *Multi-Processors*.

   Answer: On multi-core processor, the cache can be shared among the cores. On multi-processes, the cache is not shared among the processors.

4. (4pts) Do we need memory address protection in kernel mode? Explain your reason(s).

   Answer: The operating system, executing in kernel mode, is given unrestricted access to both operating-system memory and users memory. This provision allows the operating system to load users' programs into users' memory, to dump out those programs in case of errors, to access and modify parameters of system calls, to perform I/O to and from user memory, and to provide many other services.

5. (4pts) Compared with a system without dynamic loading, what is the main advantage of *dynamic loading*? Explain your reason(s).

   Answer: It has better memory-space utilization. All routines are kept on disk in a relocatable load format, only the main program is first loaded into memory and executed.

6. (4pts) Compared with compaction, what is the main advantage of *paging* to deal with external fragmentation? Explain your reason(s).

   Answer: It does not need to shuffle the memory contents to place all free memory together. It permits the physical address space of processes to be noncontiguous.

7. (4pts) Compared with local replacement (regarding frame allocation), what is the main advantage of *global replacement*? Explain your reason(s).

   Answer: It has better system throughput. A process selects a replacement frame from the set of all frames, and thus it can take a frame from another.

8. (4pts) Compared with smaller page sizes, what are the main advantages of *larger page sizes*? List two advantages; no explanation is required.

   Answer: Smaller page table size, less I/O time, and less number of page faults.

9. (4pts) Compare with special (direct) I/O instructions, what is the main advantage of *memory-mapped I/O*? Explain your reason(s).

   Answer: It is simple and faster. Device registers are mapped into the address space of the processor so that the CPU uses the standard data transfer instructions to read and write the device registers at their mapped locations in physical memory.

10. (4pts) Compared with interrupts, what is the main disadvantage of *polling* (or busy-waiting)? Explain your reason(s).

    Answer: It is inefficient. The host reads the status register over and over until the busy bit becomes clear.

3 Calculation (14pts)

1. (4pts) Given an application, 50 percent must be executed in serial, 10 percent can be executed in parallel by at most 2 processing cores, and 40 percent can be executed in parallel by at most 4 processing cores. If we run this application on a system with 4 processing cores, what is the upper bound of the speedup?

   Answer: $1/\left(0.5 + \frac{0.1}{2} + \frac{0.4}{4}\right) = \frac{1}{0.65} = \frac{20}{13}$ or $1/\left(0.5 + \frac{0.4}{4}\right) = \frac{1}{0.6} = \frac{5}{3}$.

2. (4pts) If the memory-access time is 10^{-8} second, the Translation Look-aside Buffer (TLB) access time is negligible, and the TLB hit ratio is 0.9, what is the effective memory-access time?

   Answer: $0.9 \times 10^{-8} + 0.1 \times 2 \times 10^{-8} = 1.1 \times 10^{-8}$ second.

3. (6pts) If the logical address space is 2^x bytes, a page size is 2^y bytes, and each entry in the page table is 2^z bytes, what is the condition of x, y, z to let the page table work and fit into one page?

   Answer: A page table needs $2^{(x-y)}$ entries and thus $2^{(x-y+z)}$ bytes, so the condition is $2^{(x-y+z)} \leq 2^y$, *i.e.*, $x + z \leq 2y$. An additional condition $x - y < 2^{(z+3)}$ is needed to make sure each entry in the page table is sufficiently long. The case that $x < y$ can be discussed or constrained by $x \geq y$.

4 Multithreading Model (16pts)

1. (4pts) Please describe the mechanism of executing a system call on an OS supporting kernel-level threading.

   Answer: When switching from user mode to kernel mode, the operating system can allocate on kernel thread to serve the user thread. The user thread will be blocked and the kernel thread will take over to execute the system call. When completes, the kernel thread will hand over to the user thread and be blocked.

2. A socket server in a client-server communication model should be able to serve multiple client request concurrently. Please fill the missed pseudo code of the socket servers implemented by threads-pool model. No explanation is required. Below is the `main()` function.

```
int main(){
    /* Create threads-pool */
    assert((pool = threadpool_create(THREAD, QUEUE, 0)) != NULL);

    while (true){
         /* Wait for new client request */
        newSocket = accept(welcomeSocket, (struct sockaddr *) &serverStorage, &addr_size);

        /* For each client request add the request to the queue and */
        /* assign the client request to an idle worker thread, */
        /* so the main thread can entertain next request */
        if (threadpool_add(pool, socketThread, &newSocket, 0) != 0)
            printf("Failed to add thread into pool.\n");
    }
}
```

   (a) (2pts) `threadpool_create()` function:

```
threadpool_t *threadpool_create(int thread_count, int queue_size, int flags){
    /* Allocate memory for the thread pool*/

    /* Initialize the data structure to manage threads-pool including */
    /* thread count, pointer to head, pointer to tail, status of the pool*/

    /* Allocate memory for worker threads and task queue */

    /* Initialize mutex and conditional variable first */


    /* YOUR ANSWER HERE: __________________________________________________ */

    return pool;
}
```

   Answer: `/* Creates worker threads */`

(b) (2pts) `threadpool_add()` function:

```
int threadpool_add(threadpool_t *pool, void (*function)(void *), void *argument,
  int flags){
    /* Check if the pool and function are valid and return errors if invalid */

    /* Request to lock threads-pool lock */

    /* Move the tail pointer to add new task to the queue */
    do {
        /* Is the thread queue full? Return error if full */


        /* YOUR ANSWER HERE: ________________________________________________ */

        /* pthread_cond_broadcast to wake up worker thread */
    } while(0);
    /* Unlock the mutex for modifying threads-pool data structure */
    return;
}
```

Answer: `/* Add task to queue, including thread function and arguments */`

(c) (4pts) `threadpool_thread` function:

```
/* Function for the worker thread to pick up task from task queue */
/* after receiving signals */
static void *threadpool_thread(void *threadpool) {
    for(;;) {
        /* Lock must be taken to wait on conditional variable */
        pthread_mutex_lock(&(pool->lock));

        /* Wait on condition variable, check for spurious wakeups.
        When returning from pthread_cond_wait(), we own the lock. */


        /* YOUR ANSWER HERE: ________________________________________________ */

        /* Move the pointer to the next on the queue */
        /* Unlock threads-pool */


        /* YOUR ANSWER HERE: ________________________________________________ */

        /* Unlock threads-pool */
        return(NULL);
    }
```

Answer: `/* Grab our task from thread queue */` and `/* Get to work */`

3. (4pts) Please discuss the advantages and disadvantages of using threads-pool model to implement socket servers. **Hint**: You should take into account *memory usage* and *throughput* when the average socket client request arrival rate varies.

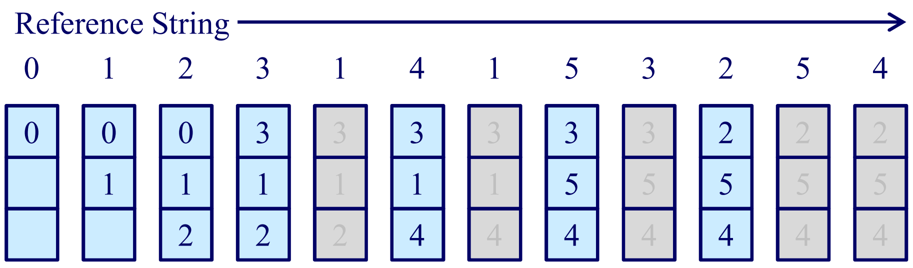
   Answer: When the arrival rate is small and the majority of the worker threads are idle, the server will "over-reserve" or over-provision the memory which will limit the memory use to other processes on the same machine, and the throughput will be same as the arrival rate. When the arrival rate is high and the majority of the worker threads are busy, the server can limit the resource used by the socket server, and the throughput will be less than the arrival rate.

5 Page Replacement Algorithms (10pts)

Given a reference string (0, 1, 2, 3, 1, 4, 1, 5, 3, 2, 5, 4), run the following page replacement algorithms, complete the figures below, and answer the number of page faults for each page replacement algorithm.

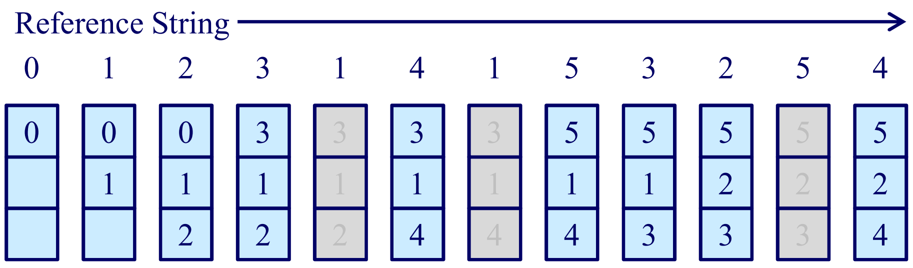
1. (4pts) Optimal page replacement. Remember to answer the number of page faults. No explanation is required.

   Answer: 7 page faults.



2. (4pts) Least Recently Used (LRU) page replacement. Remember to answer the number of page faults. No explanation is required.

   Answer: 9 page faults.



3. (2pts) What is the difficulty of using the optimal page replacement in a practical system?

   Answer: It requires future knowledge of the reference string.