Announcement

- ‘Total Score’ is updated
  - Please check it

<table>
<thead>
<tr>
<th>번호</th>
<th>학번</th>
<th>Attendance</th>
<th>PA0</th>
<th>PA1</th>
<th>PA2</th>
<th>PA3</th>
<th>PA4</th>
<th>PA5</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2753</td>
<td>8.182</td>
<td>10.000</td>
<td>9.039</td>
<td>9.000</td>
<td>18.000</td>
<td>9.667</td>
<td>63.888</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3065</td>
<td>7.273</td>
<td>10.000</td>
<td>3.013</td>
<td>8.000</td>
<td>18.135</td>
<td>0.000</td>
<td>46.422</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3814</td>
<td>8.182</td>
<td>9.000</td>
<td>8.114</td>
<td>9.056</td>
<td>18.800</td>
<td>18.073</td>
<td>71.225</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2243</td>
<td>7.273</td>
<td>10.000</td>
<td>6.011</td>
<td>9.199</td>
<td>18.189</td>
<td>0.000</td>
<td>50.672</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4275</td>
<td>8.182</td>
<td>10.000</td>
<td>9.673</td>
<td>10.000</td>
<td>20.000</td>
<td>19.997</td>
<td>77.852</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2019</td>
<td>2.727</td>
<td>10.000</td>
<td>8.000</td>
<td>9.290</td>
<td>0.000</td>
<td>0.000</td>
<td>30.018</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0197</td>
<td>6.364</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>6.364</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4601</td>
<td>8.182</td>
<td>10.000</td>
<td>0.000</td>
<td>8.120</td>
<td>0.000</td>
<td>0.000</td>
<td>26.302</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0399</td>
<td>4.545</td>
<td>10.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>14.545</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1956</td>
<td>8.182</td>
<td>10.000</td>
<td>10.000</td>
<td>10.000</td>
<td>20.000</td>
<td>20.000</td>
<td>78.182</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>2759</td>
<td>8.182</td>
<td>10.000</td>
<td>9.776</td>
<td>10.000</td>
<td>20.000</td>
<td>19.990</td>
<td>77.947</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>2341</td>
<td>8.182</td>
<td>10.000</td>
<td>9.000</td>
<td>9.600</td>
<td>18.800</td>
<td>10.000</td>
<td>65.562</td>
<td></td>
</tr>
</tbody>
</table>

- The score of PA4 will be changed!
int main (int argc, char *argv[]) {
    ...
    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    bzero((char *)&saddr, sizeof(saddr));
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = htonl(INADDR_ANY);
    saddr.sin_port = htons(port);
    bind(sockfd, (struct sockaddr *)&saddr, sizeof(saddr));

    listen(sockfd, 5);
    while (1) {
        connfd = accept(sockfd, (struct sockaddr *)&caddr, &clen);
        while ((n = read(connfd, buf, MAXLINE)) > 0) {
            printf("got %d bytes from client.\n", n);
            write(connfd, buf, n);
        }
        close(connfd);
    }
}
Iterative Servers (1)

- One request at a time

```
client 1
  call connect
  ret connect
  call read
  ret read
  close

server
  call accept
  ret accept
  write
  close

client 2
  call connect
  ret connect
  call read
  ret read
  close
```
Iterative Servers (2)

- Fundamental flaw

Client 1

- `call connect`
- `ret connect`
- `call fgets`

Server

- `call accept`
- `ret accept`
- `call read`

Client 2

- `call connect`

User goes out to lunch

Client 1 blocks waiting for user to type in data

Server blocks waiting for data from Client 1

Client 2 blocks waiting to complete its connection request until after lunch!

- Solution: use concurrent servers instead
  
  - Use multiple concurrent flows to serve multiple clients at the same time.
Concurrent Programming

Thread-based
Traditional View

- Process = process context + address space

**Process context**

- **Program context:**
  - Data registers
  - Condition codes
  - Stack pointer (SP)
  - Program counter (PC)

- **Kernel context:**
  - VM structures
  - Descriptor table
  - brk pointer

**Code, data, and stack**

- Stack
- Shared libraries
- Run-time heap
- Read/write data
- Read-only code/data

**Memory layout:**

- Stack
- Shared libraries
- Run-time heap
- Read/write data
- Read-only code/data

0
Alternate View

- Process = thread context + kernel context + address space

Thread (main thread)

- Thread context:
  - Data registers
  - Condition codes
  - Stack pointer (SP)
  - Program counter (PC)

Code and Data

- shared libraries
- run-time heap
- read/write data
- read-only code/data

Kernel context:
- VM structures
- Descriptor table
- brk pointer
A Process with Multiple Threads

- **Multiple threads can be associated with a process.**
  - Each thread has its own logical control flow (sequence of PC values)
  - Each thread shares the same code, data, and kernel context
  - Each thread has its own thread id (TID)

### Thread 1 (main thread)
- **Thread 1 context:**
  - Data registers
  - Condition codes
  - SP1
  - PC1

### Shared code and data
- **shared libraries**
- **run-time heap**
- **read/write data**
- **read-only code/data**

### Thread 2 (peer thread)
- **Thread 2 context:**
  - Data registers
  - Condition codes
  - SP2
  - PC2

### Stack
- **stack 1**
- **stack 2**
Logical View of Threads

- Threads associated with a process form a pool of peers
  - Unlike processes which form a tree hierarchy

Threads associated with process foo

Process hierarchy

shared code, data and kernel context
Threads vs. Processes

- **How threads and processes are similar**
  - Each has its own logical control flow.
  - Each can run concurrently.
  - Each is context switched.

- **How threads and processes are different**
  - Threads share code and data, processes (typically) do not.
  - Threads are somewhat less expensive than processes.
    - Linux 2.4 Kernel, 512MB RAM, 2 CPUs
      - \( \rightarrow \) 1,811 forks()/second
      - \( \rightarrow \) 227,611 threads/second (125x faster)
Pthreads Interface

- **POSIX Threads Interface**
  - Creating and reaping threads
    - `pthread_create()`
    - `pthread_join()`
  - Determining your thread ID
    - `pthread_self()`
  - Terminating threads
    - `pthread_cancel()`
    - `pthread_exit()`
    - `exit` (terminates all threads), `return` (terminates current thread)
  - Synchronizing access to shared variables
    - `pthread_mutex_init()`
    - `pthread_mutex_[un]lock()`
    - `pthread_cond_init()`
    - `pthread_cond_[timed]wait()`
    - `pthread_cond_signal()`, etc.
"hello, world" Program (1)

```c
/*
 * hello.c - Pthreads "hello, world" program
 */
#include "pthread.h"

void *thread(void *vargp);

int main() {
    pthread_t tid;
    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, NULL);
    exit(0);
}

/* thread routine */
void *thread(void *vargp) {
    printf("Hello, world!\n");
    return NULL;
}
```

Thread attributes (usually NULL)
Thread arguments (void *p)
return value (void **p)
"hello, world" Program (2)

- Execution of threaded "hello, world"

```
main thread

call pthread_create()
pthread_create() returns

call Pthread_join()

main thread waits for peer thread to terminate

pthread_join() returns

exit()

terminates main thread and any peer threads

peer thread

printf()
return NULL;
(peer thread terminates)
```
int main (int argc, char *argv[]) {
    int *connfdp;
pthread_t tid;
    . . .

    while (1) {
        connfdp = (int *)
                   malloc(sizeof(int));
        *connfdp = accept (listenfd,
                          (struct sockaddr *)&caddr,
                          &caddrlen));

        pthread_create(&tid, NULL,
                        thread_main, connfdp);
    }
}

void *thread_main(void *arg) {
    int n;
    char buf[MAXLINE];

    int connfd = *((int *)arg);
pthread_detach(pthread_self());
free(arg);

    while((n = read(connfd, buf,
                 MAXLINE)) > 0)
        write(connfd, buf, n);

    close(connfd);
return NULL;
}
Implementation Issues (1)

- **Must run “detached” to avoid memory leak.**
  - At any point in time, a thread is either *joinable* or *detached*.
  - Joinable thread can be reaped and killed by other threads
    - Must be reaped (with `pthread_join()`) to free memory resources.
  - Detached thread cannot be reaped or killed by other threads.
    - Resources are automatically reaped on termination.
    - Exit state and return value are not saved.
  - Default state is joinable.
    - Use `pthreadDetach(pthread_self())` to make detached.
Implementation Issues (2)

- Must be careful to avoid unintended sharing
  - For example, what happens if we pass the address `connfd` to the thread routine?

```c
int connfd;
...;
pthread_create(&tid, NULL, thread_main, &connfd);
...;
```

- All functions called by a thread must be thread-safe.
  - A function is said to be thread-safe or reentrant, when the function may be called by more than one thread at a time without requiring any other action on the caller’s part.
Thread-based Designs

- **Pros**
  - Easy to share data structures between threads.
    - e.g., logging information, file cache, etc.
  - Threads are more efficient than processes.

- **Cons**
  - Unintentional sharing can introduce subtle and hard-to-reproduce errors!
    - The ease with which data can be shared is both the greatest strength and the greatest weakness of threads.
Concurrent Programming

Examples
Example 1

- Fill skeleton code
  - 5 Reader + 1 Writer
  - Writer updates value 1000,000 times
  - Each reader reads value 10,000,000 times

- Use pthread_mutex
Example 2

- Implement same thing using pthread_spinlock
  - pthread_spinlock_t s
  - pthread_spin_init(&s, PTHREAD_PROCESS_PRIVATE)
  - pthread_spin_lock(&s)
  - pthread_spin_destroy(&s)

- What is the difference?
Readers-Writer Lock

- Reader blocks other reader
  - Do we need this?
- Implementation (with 2 lock)

```plaintext
Begin Read
- Lock r
- Increment b
- if b = 1, lock g
- Unlock r

End Read
- Lock r
- Decrement b
- if b = 0, unlock g
- Unlock r

Begin Write
- Lock g

End Write
- Unlock g
```
Example 3

- Implement same thing with readers-writer lock