**Problem 2: Virtual Memory (15 Pts)**

Consider the code fragment below. Suppose the resulting process is allocated \( m \) memory frames, the page size is 4K, and arrays are stored by row in the virtual address space.

```c
char A[4096][4096]; // a char is 1 byte
for (i=0; i<4096; i++)
    for (j=0; j<4096; j++)
        A[i][j] = 0;
```

a) How many page faults will this process suffer and why?

b) Replace \( A[i][j] = 0 \) with \( A[j][i] = 0 \). How many page faults will this process suffer and why?
Problem 3: Synchronization (20 Pts)

Consider a multithreaded program with $N$ threads with ids $1, 2, \ldots, N$; and each thread calls the function `ring` below with its id. Within `ring`, threads call the function `print_which_thread` in thread id order so that the output of this program would be: $1 \ 2 \ 3 \ \ldots \ N \ 1 \ 2 \ 3 \ \ldots \ N$, and so on. Complete the function `ring` below. Hint: keep track of which thread is to take its turn to call `print_which_thread`; a thread should block if it is not their turn. You must use condition variables. Try to make the program as efficient as possible! Declare all variables.

```c
void* ring (int my_id) {
    while (1) {
        print_which_thread (my_id); // just prints the single thread id, my_id
    }
}
```
**Problem 4: Network Programming (25 Pts)**

Write a simple server program that will play tic-tac-toe with a client(s) using TCP sockets. You will program the server side only. Your server will play with a given client repeatedly until the server wins the 10th game. Your server must be able to play with any number of clients but only one at a time (i.e. single-threaded is fine). You are given the following types, variables, and functions to use:

```c
Board_t B; // the board, do not define, assume this is given

// current tic-tac-toe board is in one of the following states
// 1) error (due to an illegal client move)
// 2) cwin: (client has won)
// 3) swin: (server has won)
// 4) playing: (in the middle of play)
typedef enum {playing, error, cwin, swin} board_status_t;

// Applies a move to the board and returns board_status_t
board_status apply_move (Board_t*, move_t);

// Generates next legal move for the server
move_t next_move (Board_t*);
```

The play always begins when the client sends his/her first move to the server. A move (move_t) contains a X or O and the board position (1 .. 9). The server applies the move and checks the board status to determine if the game is: a) still being played, b) if someone has won, or c) if the board is in error (due to the client move; a server move cannot produce an error). The server then sends back a response move (move_t). The response move will be the next move by the server if the game is still being played, or a special sentinel move indicating that the client has either won (a move position of -1), or the server has just won (a move position is -2), or the game is in error and will be terminated (a move position of -3).

**Things to do:**

You must define move_t such that it can be sent directly between clients and servers. Program the server including all socket setups and application protocol logic. You need not implement the functions given above, just use them. Do not worry about who is X or O.

Write your solution on the next page.
//define move_t here

void server () {

System call signatures (not all may be needed):

Sockets:
int bind (int s, struct sockaddr *addr, int len);
int accept (int s, struct sockaddr *addr, int *len);
int connect (int s, struct sockaddr *name, int len);
int listen (int s, int backlog);