Computer Networks Assignment 2

Paarth Goyal,2022343
Parth Sandeep Rastogi,2022352

Q1) Code Explanation

Server:-

```c
void get_ans_processes(char *result) {
    DIR *dir;
    struct dirent *ent;
    char path[BUFFER_SIZE], buffer[BUFFER_SIZE];
    process_info ans[2] = {{0, "", 0, 0}, {0, "", 0, 0}};

    if ((dir = opendir("/proc")) != NULL) {
        while ((ent = readdir(dir)) != NULL) {
            if (isdigit(ent->d_name[0])) {
                snprintf(path, sizeof(path), "/proc/%s/stat", ent->d_name);
                int fd = open(path, O_RDONLY);
                if (fd != -1) {
                    read(fd, buffer, sizeof(buffer) - 1);
                    close(fd);
                    int pid;
                    char pname[256];
                    long user_time, kernel_time;
                    sscanf(buffer, "%d (%[^)]) %*c %*d %*d %*d %*d %*d %*d %*d %*d %*d %*d %*d %ld %ld", &pid, pname, &user_time, &kernel_time);
                    long total_time = user_time + kernel_time;
                    if (total_time > ans[0].user_time + ans[0].kernel_time) {
                        ans[1] = ans[0];
                        ans[0] = (process_info){pid, "", user_time, kernel_time};
                        strcpy(ans[0].name, pname);
                    } else if (total_time > ans[1].user_time + ans[1].kernel_time) {
                        ans[1] = (process_info){pid, "", user_time, kernel_time};
                        strcpy(ans[1].name, pname);
                    }
                }
            }
        }
        closedir(dir);
    }
    snprintf(result, BUFFER_SIZE,
             "Top 2 CPU processes:\n"
             "A). PID: %d, Name of Proc: (%s), User Time: %ld, Kernel Time: %ld\n"
             "B). PID: %d, Name of Proc: (%s), User Time: %ld, Kernel Time: %ld\n",
             ans[0].pid, ans[0].name, ans[0].user_time, ans[0].kernel_time,
             ans[1].pid, ans[1].name, ans[1].user_time, ans[1].kernel_time);
}
```

The function get_ans_processes is the primary function which retrieves the top two cpu consuming processes from the proc directory.

The function opens the /proc directory using the “opendir” system call and reads each entry using “readdir” until it reaches the end. If the entry name starts with a digit this means that the entry corresponds to a process, after this we access the /stat directory for each process.

The entries in the stat directory are parsed in order to access the process id, process name, user time and kernel time.

The criteria for which the processes are checked for is the sum of user time and the kernel time which is the total time. We maintain an array ans of size 2 containing process info structs. The struct definition is defined as follows :

```c
typedef struct {
    int pid;
    char name[256];
    long user_time;
    long kernel_time;
} process_info;
```

We update the elements of the ans array in such a way that at the end of the pass through the proc directory we get the processes with the highest and second highest total cpu time.

Finally the process id, name, user time and kernel time for both the processes are returned as a string.

```c
int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    pthread_t thread_id;
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("Bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 10) < 0) {
        perror("Listen failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
    printf("Server listening on port %d\n", PORT);
    while (INFI) {
        new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen);
        if (new_socket < 0) {
            perror("Accept failed");
            exit(EXIT_FAILURE);
        }
        int *client_sock = malloc(sizeof(int));
        *client_sock = new_socket;
        pthread_create(&thread_id, NULL, handle_client, (void *)client_sock);
        pthread_detach(thread_id);
    }

    return 0;
```

After obtaining the process information the rest of the task was straightforward and consisted of the following steps :

1) Creating a TCP socket using the IPv4 address family
2) Setting up server address family and port (in this case 8000 was used)
3) Binding the socket with IP address and port
4) Accepting a new connection for each client inside a while loop
5) Creating a new thread to serve each client
6) Finally detaching the client thread for independent execution

```c
void *handle_client(void *client_socket) {
    int sock = *(int *)client_socket;
    free(client_socket);
    struct sockaddr_in addr;
    socklen_t addr_len = sizeof(addr);
    char buffer[BUFFER_SIZE] = {0};
    getpeername(sock, (struct sockaddr*)&addr, &addr_len);
    char *client_ip = inet_ntoa(addr.sin_addr);
    int client_port = ntohs(addr.sin_port);
    printf("Client connected: IP = %s, Port = %d\n", client_ip, client_port);
    read(sock, buffer, BUFFER_SIZE);
    char result[BUFFER_SIZE];
    get_ans_processes(result);
    send(sock, result, strlen(result), 0);
    printf("Serving client: IP = %s, Port = %d\n", client_ip, client_port);
    close(sock);
    pthread_exit(NULL);
}
```

The handle client function calls the get_ans_processes internally and then sends it over to the client, closes the socket and terminates the thread.

Client:-

```c
void *start_client(void *arg) {
    int client_fd;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE] = {0};
    if ((client_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation failed :( ");
        pthread_exit(NULL);
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("Invalid address :( \n");
        pthread_exit(NULL);
    }
    if (connect(client_fd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("Connection Error :( \n");
        pthread_exit(NULL);
    }
    send(client_fd, "GET CPU INFO", strlen("GET CPU INFO"), 0);
    printf("Request sent from thread %ld\n", pthread_self());
    read(client_fd, buffer, BUFFER_SIZE);
    printf("Received from server at thread %ld:\n%s\n", pthread_self(), buffer);
    close(client_fd);
    pthread_exit(NULL);
}

int main(int argc, char const *argv[]) {
    if (argc != 2) {
        printf("Input in wrong format buddy ");
        return -1;
    }
    int noc = atoi(argv[1]);
    pthread_t threads[noc];
    for (int i = 0; i < noc; i++) {
        pthread_create(&threads[i], NULL, start_client, NULL);
    }
    for (int i = 0; i < noc; i++) {
        pthread_join(threads[i], NULL);
    }
     printf("Connection closed\n");
    return 0;
}
```

The client side operates by executing the following steps :

1) Creates a TCP socket with IPv4 address family
2) Specifies the port number and the IP address of the server
3) Connects to the server
4) Sends a request to get the process info
5) Reads the response from the server and prints out the result

In the main function please note that the different clients are spawned as separate threads simultaneously and the number of threads that are created can be specified at the time of running the client executable.

Q2)
NOTE :- CLIENT CODE IS KEPT SAME ACROSS ALL THE QUESTIONS

1. Code Explanation

a. Server Single Thread:- Only a few changes were made to the server side which included the introduction of the get_ans_processes function(as explained above) to retrieve process information for top two cpu consuming processes and that information was sent to the clients instead of a “Hello” message which was done in the original unaltered code as per in the repository. Also modified the code so that the server does not shut down after serving one client, In the modified code the clients are served in a sequential manner and the server keeps on running so that it can serve more incoming requests.

b. Server Multithreaded:- Code same as q1

c. Server Select:- The functionality handling the clients via the select system call was already implemented in the original code, we just had to introduce the get_ans_processes function and send process data to the clients instead of echoing their message back to them. Also the number of clients that sent requests to the server were set to 100.

2. Perf Outcomes:-

Server Single Thread:-

```
Performance counter stats for './single_thread_server':

          33.94 msec task-clock                       #    0.015 CPUs utilized
             11      context-switches                 #  324.103 /sec
              7      cpu-migrations                   #  206.247 /sec
             71      page-faults                      #    2.092 K/sec
     1,78,69,805      cpu_atom/cycles/                 #    0.527 GHz                      (0.63%)
     6,57,42,239      cpu_core/cycles/                 #    1.937 GHz                      (99.26%)
     1,60,55,155      cpu_atom/instructions/           #    0.90  insn per cycle           (0.74%)
    13,87,82,837      cpu_core/instructions/           #    7.77  insn per cycle           (99.26%)
       32,53,969      cpu_atom/branches/               #   95.875 M/sec                    (0.74%)
     2,59,44,283      cpu_core/branches/               #  764.420 M/sec                    (99.26%)
          92,490      cpu_atom/branch-misses/          #    2.84% of all branches          (0.74%)
          99,411      cpu_core/branch-misses/          #    3.06% of all branches          (99.26%)
           TopdownL1 (cpu_core)                 #     31.1 %  tma_backend_bound
                                                #      3.6 %  tma_bad_speculation
                                                #     29.9 %  tma_frontend_bound
                                                #     35.4 %  tma_retiring             (99.26%)
           TopdownL1 (cpu_atom)                 #     34.6 %  tma_bad_speculation
                                                #     20.0 %  tma_retiring             (0.74%)
                                                #      7.4 %  tma_backend_bound
                                                #      7.4 %  tma_backend_bound_aux
                                                #     38.0 %  tma_frontend_bound       (0.74%)

     2.334609215 seconds time elapsed

     0.005056000 seconds user
     0.029940000 seconds sys
```

```
Performance counter stats for './single_thread_server':

         161.04 msec task-clock                       #    0.049 CPUs utilized
             19      context-switches                 #  117.980 /sec
              8      cpu-migrations                   #   49.676 /sec
             71      page-faults                      #  440.874 /sec
    30,58,70,218      cpu_atom/cycles/                 #    1.899 GHz                      (3.86%)
    34,69,34,487      cpu_core/cycles/                 #    2.154 GHz                      (94.95%)
    37,29,67,506      cpu_atom/instructions/           #    1.22  insn per cycle           (4.43%)
    71,27,05,214      cpu_core/instructions/           #    2.33  insn per cycle           (94.95%)
     6,78,21,594      cpu_atom/branches/               #  421.138 M/sec                    (4.43%)
    13,32,34,648      cpu_core/branches/               #  827.319 M/sec                    (94.95%)
        6,04,205      cpu_atom/branch-misses/          #    0.89% of all branches          (4.43%)
        4,38,174      cpu_core/branch-misses/          #    0.65% of all branches          (94.95%)
           TopdownL1 (cpu_core)                 #     33.5 %  tma_backend_bound
                                                #      3.2 %  tma_bad_speculation
                                                #     28.7 %  tma_frontend_bound
                                                #     34.6 %  tma_retiring             (94.95%)
           TopdownL1 (cpu_atom)                 #     25.2 %  tma_bad_speculation
                                                #     25.9 %  tma_retiring             (4.43%)
                                                #     26.7 %  tma_backend_bound
                                                #     26.7 %  tma_backend_bound_aux
                                                #     22.3 %  tma_frontend_bound       (4.43%)

     3.287234012 seconds time elapsed

     0.029134000 seconds user
     0.132614000 seconds sys
```

```
Performance counter stats for './single_thread_server':

         301.65 msec task-clock                       #    0.017 CPUs utilized
             47      context-switches                 #  155.809 /sec
             14      cpu-migrations                   #   46.411 /sec
             71      page-faults                      #  235.372 /sec
    22,99,38,422      cpu_atom/cycles/                 #    0.762 GHz                      (0.89%)
    62,02,80,983      cpu_core/cycles/                 #    2.056 GHz                      (98.70%)
    32,01,83,030      cpu_atom/instructions/           #    1.39  insn per cycle           (0.93%)
   1,38,24,20,861     cpu_core/instructions/           #    6.01  insn per cycle           (98.70%)
     5,48,79,943      cpu_atom/branches/               #  181.932 M/sec                    (0.97%)
    25,84,08,179      cpu_core/branches/               #  856.647 M/sec                    (98.70%)
        7,42,128      cpu_atom/branch-misses/          #    1.35% of all branches          (0.96%)
        8,30,656      cpu_core/branch-misses/          #    1.51% of all branches          (98.70%)
           TopdownL1 (cpu_core)                 #     28.4 %  tma_backend_bound
                                                #      3.6 %  tma_bad_speculation
                                                #     30.3 %  tma_frontend_bound
                                                #     37.7 %  tma_retiring             (98.70%)
           TopdownL1 (cpu_atom)                 #     16.2 %  tma_bad_speculation
                                                #     32.9 %  tma_retiring             (1.27%)
                                                #     16.5 %  tma_backend_bound
                                                #     16.5 %  tma_backend_bound_aux
                                                #     34.4 %  tma_frontend_bound       (1.30%)

    17.565722175 seconds time elapsed

     0.055823000 seconds user
     0.247218000 seconds sys
```

Server Multi Thread:-

```
Performance counter stats for './multi_thread_server':

          98.52 msec task-clock                       #    0.023 CPUs utilized
             46      context-switches                 #  466.909 /sec
              8      cpu-migrations                   #   81.202 /sec
            224      page-faults                      #    2.274 K/sec
     1,97,93,579      cpu_atom/cycles/                 #    0.201 GHz                      (62.25%)
     8,68,47,118      cpu_core/cycles/                 #    0.882 GHz                      (78.70%)
     2,42,58,545      cpu_atom/instructions/           #    1.23  insn per cycle           (75.52%)
    15,36,73,780      cpu_core/instructions/           #    7.76  insn per cycle           (78.70%)
       46,34,685      cpu_atom/branches/               #   47.043 M/sec                    (75.74%)
     2,87,37,735      cpu_core/branches/               #  291.694 M/sec                    (78.70%)
          46,898      cpu_atom/branch-misses/          #    1.01% of all branches          (79.68%)
       1,91,608      cpu_core/branch-misses/          #    4.13% of all branches          (78.70%)
          TopdownL1 (cpu_core)                 #     32.2 %  tma_backend_bound
                                               #      6.0 %  tma_bad_speculation
                                               #     30.6 %  tma_frontend_bound
                                               #     31.2 %  tma_retiring             (78.70%)
          TopdownL1 (cpu_atom)                 #     15.6 %  tma_bad_speculation
                                               #     28.9 %  tma_retiring             (79.74%)
                                               #     21.8 %  tma_backend_bound
                                               #     21.8 %  tma_backend_bound_aux
                                               #     33.7 %  tma_frontend_bound       (79.69%)

     4.321926754 seconds time elapsed

     0.017405000 seconds user
     0.080814000 seconds sys
```

```
Performance counter stats for './multi_thread_server':

         517.44 msec task-clock                       #    0.105 CPUs utilized
            186      context-switches                 #  359.463 /sec
             82      cpu-migrations                   #  158.473 /sec
            647      page-faults                      #    1.250 K/sec
    21,93,17,655      cpu_atom/cycles/                 #    0.424 GHz                      (36.12%)
    53,02,90,663      cpu_core/cycles/                 #    1.025 GHz                      (79.61%)
    24,09,55,250      cpu_atom/instructions/           #    1.10  insn per cycle           (46.13%)
    72,50,66,113      cpu_core/instructions/           #    3.31  insn per cycle           (79.61%)
     4,50,05,105      cpu_atom/branches/               #   86.977 M/sec                    (47.45%)
    13,56,98,348      cpu_core/branches/               #  262.250 M/sec                    (79.61%)
        4,50,105      cpu_atom/branch-misses/          #    1.00% of all branches          (48.63%)
        8,42,514      cpu_core/branch-misses/          #    1.87% of all branches          (79.61%)
          TopdownL1 (cpu_core)                 #     24.7 %  tma_backend_bound
                                               #      5.2 %  tma_bad_speculation
                                               #     34.6 %  tma_frontend_bound
                                               #     35.5 %  tma_retiring             (79.61%)
          TopdownL1 (cpu_atom)                 #     12.5 %  tma_bad_speculation
                                               #     28.7 %  tma_retiring             (49.99%)
                                               #     26.3 %  tma_backend_bound
                                               #     26.3 %  tma_backend_bound_aux
                                               #     32.5 %  tma_frontend_bound       (50.49%)

     4.949620225 seconds time elapsed

     0.084522000 seconds user
     0.427794000 seconds sys
```

```
Performance counter stats for './multi_thread_server':

       1,069.55 msec task-clock                       #    0.167 CPUs utilized
            438      context-switches                 #  409.517 /sec
            168      cpu-migrations                   #  157.075 /sec
          1,094      page-faults                      #    1.023 K/sec
    36,78,85,909      cpu_atom/cycles/                 #    0.344 GHz                      (52.65%)
    95,93,87,102      cpu_core/cycles/                 #    0.897 GHz                      (88.49%)
    41,70,02,478      cpu_atom/instructions/           #    1.13  insn per cycle           (60.71%)
 1,26,88,12,846      cpu_core/instructions/           #    3.45  insn per cycle           (88.49%)
     7,84,54,527      cpu_atom/branches/               #   73.353 M/sec                    (61.52%)
    23,74,39,633      cpu_core/branches/               #  221.999 M/sec                    (88.49%)
        7,80,504      cpu_atom/branch-misses/          #    0.99% of all branches          (59.44%)
       14,69,838      cpu_core/branch-misses/          #    1.87% of all branches          (88.49%)
          TopdownL1 (cpu_core)                 #     29.0 %  tma_backend_bound
                                               #      5.0 %  tma_bad_speculation
                                               #     33.6 %  tma_frontend_bound
                                               #     32.4 %  tma_retiring             (88.49%)
          TopdownL1 (cpu_atom)                 #     16.6 %  tma_bad_speculation
                                               #     26.7 %  tma_retiring             (58.51%)
                                               #     23.5 %  tma_backend_bound
                                               #     23.5 %  tma_backend_bound_aux
                                               #     33.2 %  tma_frontend_bound       (59.57%)

     6.410059061 seconds time elapsed

     0.202649000 seconds user
     0.854548000 seconds sys
```

Server Select:-

```
Performance counter stats for './select_server':

          39.82 msec task-clock                       #    0.013 CPUs utilized
              7      context-switches                 #  175.790 /sec
              3      cpu-migrations                   #   75.339 /sec
             69      page-faults                      #    1.733 K/sec
     3,08,27,938      cpu_atom/cycles/                 #    0.774 GHz                      (4.44%)
     7,96,14,481      cpu_core/cycles/                 #    1.999 GHz                      (90.55%)
     4,88,50,046      cpu_atom/instructions/           #    1.58  insn per cycle           (6.94%)
    14,74,50,223      cpu_core/instructions/           #    4.78  insn per cycle           (90.55%)
       89,21,718      cpu_atom/branches/               #  224.050 M/sec                    (6.94%)
     2,75,44,537      cpu_core/branches/               #  691.721 M/sec                    (90.55%)
        88,533      cpu_atom/branch-misses/          #    0.99% of all branches          (8.11%)
     1,13,755      cpu_core/branch-misses/          #    1.28% of all branches          (90.55%)
        TopdownL1 (cpu_core)                 #     33.1 %  tma_backend_bound
                                             #      4.0 %  tma_bad_speculation
                                             #     29.0 %  tma_frontend_bound
                                             #     33.9 %  tma_retiring             (90.55%)
        TopdownL1 (cpu_atom)                 #     14.3 %  tma_bad_speculation
                                             #     37.0 %  tma_retiring             (9.45%)
                                             #     18.6 %  tma_backend_bound
                                             #     18.6 %  tma_backend_bound_aux
                                             #     30.1 %  tma_frontend_bound       (9.45%)

     3.170204752 seconds time elapsed

     0.007089000 seconds user
     0.033422000 seconds sys
```

```
Performance counter stats for './select_server':

         101.79 msec task-clock                       #    0.037 CPUs utilized
             21      context-switches                 #  206.305 /sec
              2      cpu-migrations                   #   19.648 /sec
             68      page-faults                      #  668.037 /sec
     8,05,12,952      cpu_atom/cycles/                 #    0.791 GHz                      (10.19%)
    34,87,05,478      cpu_core/cycles/                 #    3.426 GHz                      (85.89%)
    11,16,17,050      cpu_atom/instructions/           #    1.39  insn per cycle           (12.15%)
    78,31,78,768      cpu_core/instructions/           #    9.73  insn per cycle           (85.89%)
     2,05,28,673      cpu_atom/branches/               #  201.675 M/sec                    (12.14%)
    14,63,26,644      cpu_core/branches/               #    1.438 G/sec                    (85.89%)
        1,57,331      cpu_atom/branch-misses/          #    0.77% of all branches          (12.14%)
        4,18,010      cpu_core/branch-misses/          #    2.04% of all branches          (85.89%)
        TopdownL1 (cpu_core)                 #     29.8 %  tma_backend_bound
                                             #      3.2 %  tma_bad_speculation
                                             #     29.4 %  tma_frontend_bound
                                             #     37.6 %  tma_retiring             (85.89%)
        TopdownL1 (cpu_atom)                 #     14.1 %  tma_bad_speculation
                                             #     32.4 %  tma_retiring             (12.15%)
                                             #     17.1 %  tma_backend_bound
                                             #     17.1 %  tma_backend_bound_aux
                                             #     36.4 %  tma_frontend_bound       (12.14%)

     2.724236453 seconds time elapsed

     0.016275000 seconds user
     0.086464000 seconds sys
```

```
Performance counter stats for './select_server':

         159.04 msec task-clock                       #    0.060 CPUs utilized
             35      context-switches                 #  220.064 /sec
             12      cpu-migrations                   #   75.450 /sec
             70      page-faults                      #  440.128 /sec
    12,80,28,136      cpu_atom/cycles/                 #    0.805 GHz                      (3.86%)
    64,90,04,168      cpu_core/cycles/                 #    4.081 GHz                      (94.88%)
    15,51,76,351      cpu_atom/instructions/           #    1.21  insn per cycle           (4.48%)
 1,44,08,47,281      cpu_core/instructions/           #   11.25  insn per cycle           (94.88%)
     2,84,53,638      cpu_atom/branches/               #  178.903 M/sec                    (4.49%)
    26,91,75,648      cpu_core/branches/               #    1.692 G/sec                    (94.88%)
        3,17,696      cpu_atom/branch-misses/          #    1.12% of all branches          (4.49%)
        7,95,363      cpu_core/branch-misses/          #    2.80% of all branches          (94.88%)
        TopdownL1 (cpu_core)                 #     29.1 %  tma_backend_bound
                                             #      3.6 %  tma_bad_speculation
                                             #     30.3 %  tma_frontend_bound
                                             #     37.0 %  tma_retiring             (94.88%)
        TopdownL1 (cpu_atom)                 #     22.9 %  tma_bad_speculation
                                             #     27.8 %  tma_retiring             (4.57%)
                                             #     18.0 %  tma_backend_bound
                                             #     18.0 %  tma_backend_bound_aux
                                             #     31.3 %  tma_frontend_bound       (4.49%)

     2.669352591 seconds time elapsed

     0.031764000 seconds user
     0.128050000 seconds sys
```

3. Results Interpretation:-

Context Switches

- Single-threaded
  - 10 clients: 11 context switches
  - 50 clients: 19 context switches
  - 100 clients: 47 context switches

- Multi-threaded
  - 10 clients: 48 context switches
  - 50 clients: 186 context switches
  - 100 clients: 438 context switches

- Select
  - 10 clients: 7 context switches
  - 50 clients: 21 context switches
  - 100 clients: 35 context switches

Interpretation

The multi-threaded approach shows the highest number of context switches,which increases as the client request increases . This is caused due to the overhead of managing multiple threads, which many times leads to frequent context switching as the scheduler allocates CPU time among the threads. Single-threaded applications show moderate increases, while select-based implementations maintain the lowest context switch count, suggesting that they handle concurrent connections efficiently without much context switching between kernel and user space .

Page Faults

- Single-threaded
  - 10 clients: 71 page faults
  - 50 clients: 71 page faults
  - 100 clients: 71 page faults

- Multi-threaded
  - 10 clients: 224 page faults

  - 50 clients: 647 page faults
  - 100 clients: 1094 page faults

- Select
  - 10 clients: 69 page faults
  - 50 clients: 68 page faults
  - 100 clients: 70 page faults

Interpretation
Single-threaded and select implementations maintain consistent and also almost same number of page fault counts, indicating stable memory usage regardless of client load . Reason for which can be that single threaded server is handling client sequentially and it need not allocate more dynamic memory hence less page fault and as for select it monitors all the client sockets which are ready for i/o and then sequentially processes them, due to this sequential one at a time handling there are lesser page faults. In contrast, the multi-threaded approach shows a drastic increase in page faults as client requests increase, suggesting that each thread may be accessing memory in a less efficient manner and also creating threads will take up more space thus causing more page faults and potential memory thrashing.

CPU Migration

- Single-threaded
  - 10 clients: 7 migration
  - 50 clients: 8 migration
  - 100 clients: 14 migration

- Multi-threaded
  - 10 clients: 8 migration
  - 50 clients: 82 migration
  - 100 clients: 168 migration

- Select
  - 10 clients: 3 migration
  - 50 clients: 2 migration
  - 100 clients: 12 migration

Interpretation
The analysis of CPU migration across different connection handling models shows significant differences in resource management and performance efficiency. In the

single-threaded approach, CPU migrations remain low and stable, indicating effective CPU utilization even as client requests increase. This stability is contrasted starkly by the multi-threaded model, which experiences a substantial rise in migrations, particularly under higher loads, suggesting that the scheduler frequently shifts threads among CPUs to manage load. This increased context switching can lead to higher overhead and bad performance. In comparison, the select model consistently shows the lowest migration counts across all client loads, demonstrating its efficiency in managing concurrent connections with minimal CPU disruption.

Task Clock (in ms) and CPU Utilized
- Single-threaded
  - 10 clients: 33.94 ms and 0.015 cpu
  - 50 clients: 161.04 ms and 0.049 cpu
  - 100 clients: 301.65 ms and 0.017 cpu

- Multi-threaded
  - 10 clients: 98.52 ms and 0.023 cpu
  - 50 clients: 517.44 ms and 0.105 cpu
  - 100 clients: 1069.55 ms and 0.167 cpu

- Select
  - 10 clients: 39.82 ms and 0.013 cpu
  - 50 clients: 101.79 ms and 0.037 cpu
  - 100 clients: 159.04 ms and 0.068 cpu

Interpretation

The multi-threaded server consumes the most CPU time, reflecting its increased context switches and page faults. As client requests grow, the task clock for multi-threaded implementations rises sharply, indicating a higher processing overhead. A reason for this can be that each thread goes to the processor to complete and remains there for a while before getting freed. Single-threaded implementations show a more gradual increase in task clock time, while select-based servers maintain the lowest CPU usage as they optimize CPU time by monitoring and selecting only those client connections which are ready for i/o, this results in no wastage of cpu resources on idling (when there are no read or write operations) as connections are processed right away without allowing any single connection to block.

Overall Observations and Reasoning

1. Multi-threaded Performance

- High context switches and page faults indicate significant overhead in managing multiple threads. This is likely due to contention for shared resources and increased memory access patterns, which lead to inefficient memory use.

2. Single-threaded and Select Implementations

- These approaches handle concurrent connections more efficiently, showing stable performance metrics. Their lower context switches and consistent page faults imply that they are better at managing limited resources without excessive overhead.
- Another reason for single thread and select having similar stats is that Select is more efficiently managed sequential handling only with select eliminating idle time and wastage of resources.

Conclusion

The results highlight the trade-offs between different connection handling models. While multi-threaded implementations may offer parallelism, they incur significant overhead that can lead to inefficiencies. Conversely, single-threaded and select-based models provide more stable performance, making them suitable for applications where resource management and efficiency are critical.

How to run

:- cd to the directory of code
:- run make
:- one terminal do taskset -c 1 ./multi_thread_server
or taskset -c 1 ./single_thread_server
or taskset -c 1 ./select_server
:- on other terminal do taskset -c 2 ./client <num of clients>

REFERENCES :

1. readdir(3) - Linux manual page (man7.org)
2. opendir(3) - Linux manual page (man7.org)
3. inet_ntoa(3): Internet address change routines - Linux man page (die.net)
4. c - Getting IPV4 address from a sockaddr structure - Stack Overflow
5. android - what are the meaning of values at proc/[pid]/stat? - Stack Overflow
6. c - htons() function in socket programing - Stack Overflow
7. getpeername(2) - Linux manual page (man7.org)
8. The Pthreads Library - Multithreaded Programming Guide (oracle.com)
9. socket_programming/select/server.cpp at main · AkankshaSingal8/socket_programming (github.com)

Github Repository Link :

https://github.com/parthrastogicoder/CN_assignment_2