**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**\*\*\*\*\*\*\*\***

A picture containing icon

Description automatically generated

**BÁO CÁO**

**Bài tập thực hành :**

**Chương 4: Socket APIs**

**Học phần: Thực hành Lập Trình Mạng**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn** | : Trần Hải Anh |
| **Mã lớp** | : 151907 |
| **Sinh viên thực hiện** | : Phạm Vân Anh |
| **Mã số sinh viên** | : 20214988 |

**Hà Nội, tháng 9 năm 2024**

**Bài 1:** Xử lý tiến trình zombie

**Yêu cầu 1:** Yêu cầu 1: Hãy viết một chương trình đơn giản sử dụng hàm fork() để tạo tiến trình con mà không dùng trình xử lý tín hiệu (signal handler). Trong đó tiến trình cha tạo ra một tiến trình con bằng cách sử dụng fork(). Tiến trình cha gọi hàm sleep(30); để ngủ 30s và không chờ tiến trình con kết thúc, điều này có thể dẫn đến tiến trình con trở thành zombie.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

pid\_t pid = fork();

if (pid < 0) {

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid == 0) {

printf("This is the child process. My PID is %d.\n", getpid());

exit(0);

} else {

printf("This is the parent process. My child's PID is %d.\n", pid);

printf("My PID is %d.\n", getpid());

wait(NULL);

printf("Parent process has collected the zombie process.\n");

sleep(30);

}

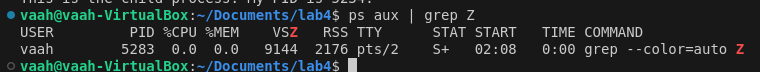
return 0;

}

Câu hỏi 1: Kết quả thu được là gì? Tiến trình nào đang ở trạng thái Z? Từ khóa nghĩa là gì?

A screenshot of a computer

Description automatically generated

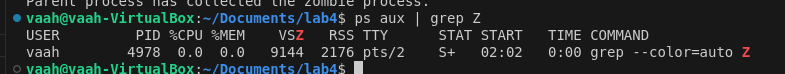


* **Từ khóa <defunct> nghĩa là gì**: <defunct> là từ dùng để chỉ các tiến trình đã kết thúc nhưng vẫn còn tồn tại trong bảng tiến trình của hệ điều hành. Tiến trình đó trở thành zombie vì tiến trình cha chưa thu dọn nó bằng cách gọi wait() hoặc waitpid().

**Yêu cầu 2:** Làm lại các bước ở trên và cho biết có còn tiến trình zombie nào xuất hiện không? vì sao?

A screenshot of a computer

Description automatically generated



* **Kết quả**: Sau khi sửa đổi và thêm wait(), khi chạy lại chương trình và kiểm tra bằng lệnh ps aux | grep Z, sẽ không còn xuất hiện bất kỳ tiến trình zombie nào nữa.
* **Giải thích**: Việc sử dụng wait() trong tiến trình cha sẽ giúp tiến trình cha thu gom tiến trình con sau khi nó kết thúc, ngăn chặn tiến trình con trở thành zombie.

**Bài 2: Viết một chương trình client-server**

**-** Server là một chương trình đa tiến trình, có thể nhận kết nối với nhiều client cùng lúc, với mỗi client thì server sẽ dùng hàm fork() để sinh ra một tiến trình con để trao đổi thông tin với client đó. Server có kiểm soát thu hồi tiến trình zombie client.

- Server có lưu trữ 10 câu hỏi trắc nghiệm, mỗi câu hỏi trắc nghiệm có 4 đáp án, trong đó có một đáp án đúng.

- Sau khi một client liên kết với server thì sẽ nhận được lần lượt các câu hỏi kèm theo 4 câu trả lời theo thứ tự ngẫu nhiên.

- Client nhận được câu hỏi và danh sách 4 câu trả lời thì sẽ cho người dùng gõ vào câu trả lời và gửi lại cho server. Mỗi câu trả lời đúng thì client đó được 1 điểm.

- Sau khi trả lời xong 10 câu hỏi thì chương trình sẽ gửi lại điểm của client trong lần chơi đó.

* **Server:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <signal.h>

#include <sys/wait.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

#define MAX\_QUESTIONS 10

char \*questions[MAX\_QUESTIONS] = {

"1 + 1 = ?\nA. 1\nB. 2\nC. 3\nD. 4",

"What is the capital of France?\nA. London\nB. Paris\nC. Berlin\nD. Madrid",

"5 \* 5 = ?\nA. 20\nB. 25\nC. 30\nD. 35",

"What is the largest planet?\nA. Earth\nB. Mars\nC. Jupiter\nD. Venus",

"What is the boiling point of water?\nA. 90°C\nB. 100°C\nC. 110°C\nD. 120°C",

"Which language is used for web development?\nA. Python\nB. HTML\nC. C++\nD. Java",

"Who developed the theory of relativity?\nA. Newton\nB. Einstein\nC. Galileo\nD. Bohr",

"What is the capital of Japan?\nA. Seoul\nB. Tokyo\nC. Kyoto\nD. Osaka",

"What is 10 / 2 = ?\nA. 3\nB. 5\nC. 7\nD. 9",

"Which element is denoted by 'O' in chemistry?\nA. Oxygen\nB. Hydrogen\nC. Helium\nD. Carbon"

};

char correct\_answers[MAX\_QUESTIONS] = {'B', 'B', 'B', 'C', 'B', 'B', 'B', 'B', 'B', 'A'};

// Signal handler to prevent zombie processes

void sigchld\_handler(int sig) {

(void)sig;

while (waitpid(-1, NULL, WNOHANG) > 0);

}

void handle\_client(int connfd) {

char buffer[BUFFER\_SIZE];

int score = 0;

for (int i = 0; i < MAX\_QUESTIONS; i++) {

if (questions[i] != NULL) {

send(connfd, questions[i], strlen(questions[i]), 0);

} else {

snprintf(buffer, sizeof(buffer), "Question %d does not exist.\n", i+1);

send(connfd, buffer, strlen(buffer), 0);

}

memset(buffer, 0, BUFFER\_SIZE);

read(connfd, buffer, sizeof(buffer));

if (buffer[0] == correct\_answers[i]) {

score++;

}

}

snprintf(buffer, sizeof(buffer), "Your final score is: %d/%d\n", score, MAX\_QUESTIONS);

send(connfd, buffer, strlen(buffer), 0);

printf("Client's score: %d\n", score);

close(connfd);

exit(0);

}

int main() {

int listenfd, connfd;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t client\_len = sizeof(client\_addr);

pid\_t pid;

// Create a socket

if ((listenfd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

// Define server address

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket

if (bind(listenfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Bind failed");

close(listenfd);

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(listenfd, 5) < 0) {

perror("Listen failed");

close(listenfd);

exit(EXIT\_FAILURE);

}

// Handle SIGCHLD to prevent zombie processes

signal(SIGCHLD, sigchld\_handler);

printf("Server is running on port %d...\n", PORT);

while (1) {

// Accept incoming connection

if ((connfd = accept(listenfd, (struct sockaddr \*)&client\_addr, &client\_len)) < 0) {

perror("Accept failed");

continue;

}

// Fork a child process to handle the client

if ((pid = fork()) == 0) {

close(listenfd);

handle\_client(connfd);

} else if (pid > 0) {

close(connfd);

} else {

perror("Fork failed");

close(connfd);

}

}

return 0;

}

* **Client**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

#define MAX\_QUESTIONS 10

int main() {

int client\_socket;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

int n;

// Create socket

if ((client\_socket = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Define server address

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr) <= 0) {

perror("Invalid address or Address not supported");

close(client\_socket);

exit(EXIT\_FAILURE);

}

// Connect to server

if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Connection failed");

close(client\_socket);

exit(EXIT\_FAILURE);

}

for (int i = 0; i < MAX\_QUESTIONS; i++) {

memset(buffer, 0, BUFFER\_SIZE);

read(client\_socket, buffer, sizeof(buffer));

printf("Question %d: %s\n", i + 1, buffer);

while (1) {

printf("Enter your answer (A, B, C, or D): ");

fgets(buffer, BUFFER\_SIZE, stdin);

if (buffer[strlen(buffer) - 1] == '\n') {

buffer[strlen(buffer) - 1] = '\0';

}

if (buffer[0] == 'A' || buffer[0] == 'B' || buffer[0] == 'C' || buffer[0] == 'D') {

send(client\_socket, buffer, strlen(buffer), 0);

break;

} else {

printf("Invalid answer. Please enter A, B, C, or D.\n");

}

}

}

// Receive and print the final score

memset(buffer, 0, BUFFER\_SIZE);

read(client\_socket, buffer, sizeof(buffer));

printf("%s", buffer);

close(client\_socket);

return 0;

}

* **Client 1:**

**A screenshot of a computer

Description automatically generated**

* **Client 2:**

**A screenshot of a computer

Description automatically generated**

* **Server sau khi hoàn thành câu hỏi:**

**A screen shot of a computer

Description automatically generated**