Звіт

Лабораторна робота №5

студента групи ТТП-32

Ходакова Максима

**Варіант 10**

**Умова**

Запрограмувати взаємодію 2 потоків, що реалізують деякі функції int f(int x) {...} та int g(int x) {...}  
Вміст функцій f і g треба буде змінити при здачі перед запуском на вимогу викладача.  
Врахувати, що функція може повертати:  
1) 0 (трактується як false для логічних операцій)  
2) інше ціле число (трактується як true для логічних операцій)  
3) **нічого не повертати, зациклюватись** (можна змоделювати нескінченим циклом while(1); або sleep(10...0); )  
Тобто функції f та g можуть бути **частково визначені** (тобто «зациклюватись» і ніколи не повертати результат). Потрібно *коректно опрацювати* таку ситуацію і *запитати користувача*: «1) продовжити обчислення, 2) припинити або 3) продовжити, не перепитуючи більше» наприклад, кожні 10 секунд.  
  
Також врахувати, що функції f(x) і g(x) є "**чистими функціями**" (<https://en.wikipedia.org/wiki/Pure_function>), тобто вони займаються тільки обчисленням значення над вхідним аргументом, вони **не обробляють ніяких інших запитів** (у тому числі – про завершення обчислень) і **не взаємодіють з іншими процесами та потоками** ні в який інший спосіб, окрім викликів обчислень f(x) і g(x) (тобто запуску функції на обчислення) та повернення результату (коли обчислення результату завершено) через return(...);.  
  
Потрібно обчислити деякий результат над f і g (**згідно варіанту нижче, за номером у списку групи**), враховуючи, що f і g можуть нічого не повернути (зависнути, невизначеність), а також спираючись на наступні **правила обчислень (логіка Кліні)** (<https://en.wikipedia.org/wiki/Three-valued_logic#Kleene_and_Priest_logics>):  
1) x && false == false && x == false  
2) x || true == true || x == true  
3) 0 \* x == x \* 0 == 0 для довільного числа x  
Основна ідея – "**ліниві**" симетричні (комутативні) **обчислення**: зупинятись і видавати результат, як тільки є найменші підстави це зробити, тобто як тільки з якихось компонентів обчислення стає зрозуміло, що результат вже визначено і він надалі не зміниться (бо не залежить від недообчислених компонентів виразу).

10. **Взаємодія процесів. Паралелізм. Обмін повідомленнями**. Обчислити f(x) \* g(x), використовуючи 2 допоміжні процеси: один обчислює f(x), а інший – g(x). Основна програма виконує ввод-вивід та операцію \*. Використати обмін повідомленнями між процесами (Messages). Реалізувати варіант **блокуючих** операцій обміну повідомленнями, тобто з очікуванням обробки повідомлення і відповіді на повідомлення (і “зависанням” процесу на цей час). Функції f(x) та g(x) “нічого не знають друг про друга” і не можуть комунікувати між собою.

**Реалізація (MacOS Unix-like система)**

#include **<sys/msg.h>**#include **<sys/ipc.h>**#include **<unistd.h>**#include **<stdio.h>**#include **<stdlib.h>***// Оголошення структури повідомлень***struct** msgbuf {  
 **long** mtype; *// тип повідомлення* **int** mvalue; *// значення, що передається в повідомленні*};  
  
*// Функція, що відправляє результат обчислення в чергу повідомлень***void** send\_result(**int** msgid, **int** result, **int** result\_type) {  
 **struct** msgbuf msg;  
 msg.mtype = result\_type;  
 msg.mvalue = result;  
 msgsnd(msgid, &msg, **sizeof**(msg.mvalue), 0);  
}  
  
*// Функція f(x), яка тепер повертає значення x замість вічного циклу***int** f(**int** x) {  
 *// Імітація тривалої операції* sleep(1);  
 **return** x \* 2; *// Приклад обчислення*}  
  
*// Функція g(x), що затримується на 3 секунди***int** g(**int** x) {  
 sleep(3);  
 **return** x \* 3; *// Приклад обчислення*}  
  
**int** main(**int** argc, **char** \*argv[]) {  
 **if** (argc != 2) {  
 printf(**"Usage: %s <integer value>\n"**, argv[0]);  
 **return** -1;  
 }  
  
 **int** x = atoi(argv[1]);  
  
 *// Створення черги повідомлень* **int** msgid = msgget(**IPC\_PRIVATE**, 0666 | **IPC\_CREAT**);  
  
 **if** (fork() == 0) {  
 *// Дочірній процес для f(x)* **int** result = f(x);  
 send\_result(msgid, result, 1);  
 exit(0);  
 }  
  
 **if** (fork() == 0) {  
 *// Дочірній процес для g(x)* **int** result = g(x);  
 send\_result(msgid, result, 2);  
 exit(0);  
 }  
  
 *// Очікування результатів від обох процесів* **struct** msgbuf msg;  
 **int** results[2] = {0}, count = 0;  
 **while** (count < 2) {  
 **if** (msgrcv(msgid, &msg, **sizeof**(msg.mvalue), 0, 0) > 0) {  
 results[msg.mtype - 1] = msg.mvalue;  
 count++;  
 }  
 }  
  
 *// Обчислення та виведення результату* **int** product = results[0] \* results[1];  
 printf(**"Result: %d\n"**, product);  
  
 *// Видалення черги повідомлень* msgctl(msgid, **IPC\_RMID**, **NULL**);  
  
 **return** 0;  
}

**Тестування**

**A screenshot of a computer

Description automatically generated**

**Опис програми**

Ця програма використовує міжпроцесне спілкування через черги повідомлень в UNIX-подібних системах для обчислення виразу f(x) \* g(x), де f(x) та g(x) — деякі функції, задані користувачем. Робота програми поділяється на декілька етапів:

1. Вхідні дані: Програма приймає один аргумент командного рядка ціле число x, яке використовується як вхідний параметр для функцій f(x) та g(x).
2. Створення черги повідомлень: Через msgget, програма створює чергу повідомлень з унікальним ключем (тут використовується IPC\_PRIVATE для автоматичного генерування ключа). Ця черга буде використовуватися для обміну результатами між процесами, які виконують f(x) та g(x).
3. Паралельне виконання f(x) та g(x): Програма створює два дочірніх процеси за допомогою fork(). Кожен дочірній процес виконує одну з функцій: Перший дочірній процес викликає f(x), що імітує тривалу операцію (через sleep(1)) і повертає x \* 2. Другий дочірній процес викликає g(x), що також імітує тривалу операцію (через sleep(3)) і повертає x \* 3. Після обчислення кожен дочірній процес використовує send\_result, щоб відправити результат своєї функції назад до основного процесу через чергу повідомлень. Кожен результат має унікальний mtype (1 або 2), що дозволяє основному процесу розрізняти результати f(x) та g(x).
4. Очікування та обробка результатів: Основний процес чекає на повідомлення з результатами обчислень f(x) та g(x), використовуючи msgrcv. Він читає повідомлення до тих пір, поки не отримає обидва результати.
5. Обчислення кінцевого результату: Після отримання обох результатів основний процес перемножує їх і виводить результат на екран.
6. Прибирання: На завершення, програма видаляє чергу повідомлень за допомогою msgctl(msgid, IPC\_RMID, NULL), очищаючи усі системні ресурси, асоційовані з чергою.

**Висновок**

Побудована програма яка ілюструє принцип паралельного виконання задач і міжпроцесного спілкування в операційних системах UNIX-подібного типу. Використання черг повідомлень дозволяє безпечно та ефективно обмінюватися даними між процесами, що виконуються паралельно. GitHub: [лінк](https://github.com/maksymkhodakov/OSLab5).