ACS. OS. IHW3

Горбачева Маргарита Валерьевна | БПИ-245

Задание: **Общее условие**

`Разработать программы, состоящие из нескольких взаимодействующих параллельных процессов. Количество разрабатываемых программ и их характеристики определяются в соответствии с требованиями выполнения на соответствующую оценку.`

Задание: индивидуальный вариант 23

`Задача о болтунах. N болтунов имеют телефоны. Они либо некоторое (случайное) время ждут звонков, либо звонят друг другу, чтобы побеседовать. Если телефон случайного абонента занят, болтун будет звонить другому случайному абоненту, пока ему кто-нибудь не ответит. Побеседовав некоторое время, болтун или ждет звонка, или звонит на другой случайный номер. Создать многопроцессное приложение, моделирующее поведение болтунов. Каждый болтун моделируется отдельным процессом.`

Решение данного домашнего задания лежит на GitHub по ссылке: https://github.com/Misss-Lacoste/ACS-OS/tree/main/IHW3

NotaBene!! *Прошу заметить, что комментарии в кодах написаны на английском языке, т.к. задание выполнялось на виртуальной машине линукса, которая не поддерживает русский язык.*

Структура работы:

Данное домашнее задание разделено на три подпапочки с кодами.

1. Подпапочка grade4-6
2. Подпапочка grade7-8
3. Подпапочка grade9

Инструкция по запуску программ:

1. В первой подпапочке на оценку 4-6 находится один кодовый файл - `main.c`. Открываем терминал и компилируем стандартно (но с особенными флажками: флажки -lrt и -lpthread — это библиотеки линковки для компилятора, подробности все не знаю, без них не собирались программы, stackoverflow со своими советами пришел на помощь):
    - gcc -o main main.c -lrt -lpthread

- и теперь ./main n (где n - количество болтунов. Я, например, тестировала с количеством 3, потому что мне нравится это число, а вообще по коду предусмотрено до 20 болтунов). NB! нумерация болтунов [0, n-1], прям по канонам cpp. Любуемся консольным выводом.

2. Во второй подпапочке на оценку 7-8 находится также один кодовый файл - `main2.c` и файл скрипта - `run2.sh`. Запустить можно двумя способами:
   а) *Вручную*. В таком случае нам необходимо будет сплитануть терминал на (n+1) терминал, то есть - основной, где происходит компиляция, и еще столько терминалов, сколько болтунов вы хотите помучить. Таким образом, для теста трех болтунов будет 4 терминала.

   а.1) В первом терминале стандартная компиляция с особенными флажками и запуск программы:
   - gcc -o main2 main2.c -lrt -lpthread
   - ./main2 controller n (где n - количество болтунов). NB! нумерация болтунов [0, n-1].

   а.2) Далее открываем еще три терминала и в каждом из них запускаем программы стандартным способом, но указываем, что запускаем персон-болтунов, добавляя слово ‘person’, и после пишем id болтуна и следующим числом количество возможных болтунов, сколько мы изначально захотели:

   ./main2 person 0 3
   ./main2 person 1 3
   ./main2 person 2 3

   б) Запустив скрипт. Изначально нужно сделать скрипт исполняемым посредством команды chmod +x run2.sh, то есть даем права на исполнение. И далее запускаем ./run2.sh. Однако вывод будет отличаться, в данном случае будет каша в одном терминале, не очень удобно и красиво, но для ленивого настроения.

3. В третьей подпапочке лежит два кодовых файла - `main3.c` и `observer.c` и скрипт - `run3.sh`. Аналогично, запустить можно двумя способами:
   а) Вручную. Также необходимо будет сплитануть терминал, но уже на (n+2) терминалов. Таким образом, для 3х болтунов будет 5 терминалов.

   а.1) В первом терминале стандартная компиляция двух программ (файлов)
   - gcc -o main3 main3.c -lrt -lpthread
   - gcc -o observer observer.c -lrt -lpthread

   а.2) Теперь в новом втором терминале

./main3 controller n (где n аналогично - количество болтунов)

а.3) Теперь в каждом остальном терминале [3, n+2]. Также помним про слово ‘person’, которое нужно добавить в команду запуска:

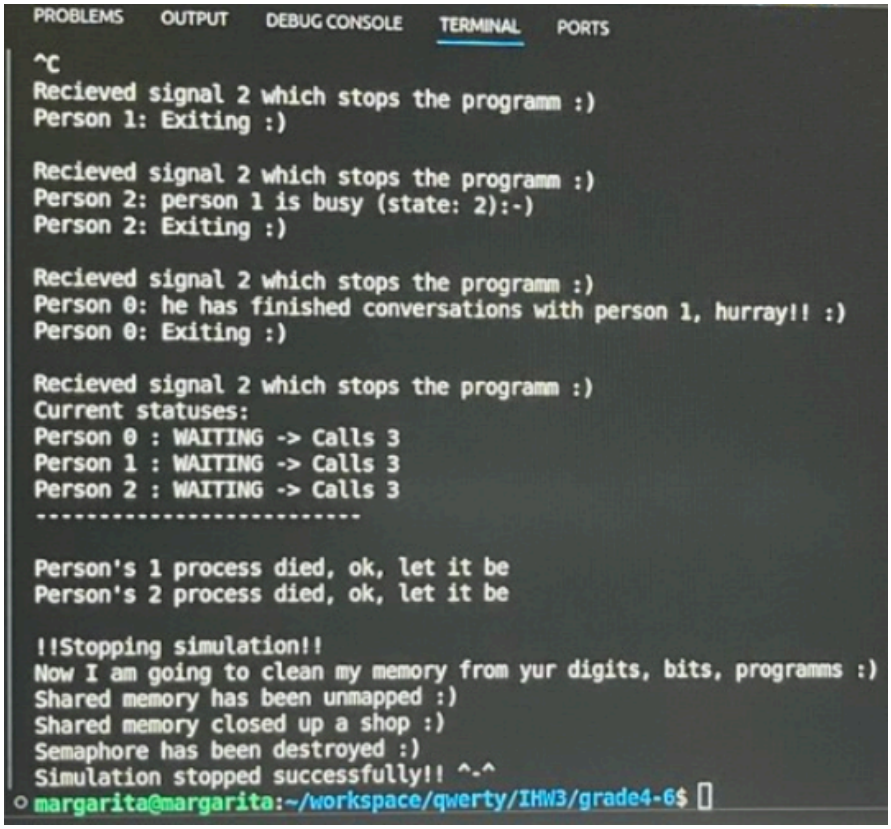
./main3 person 0 3

./main3 person 1 3

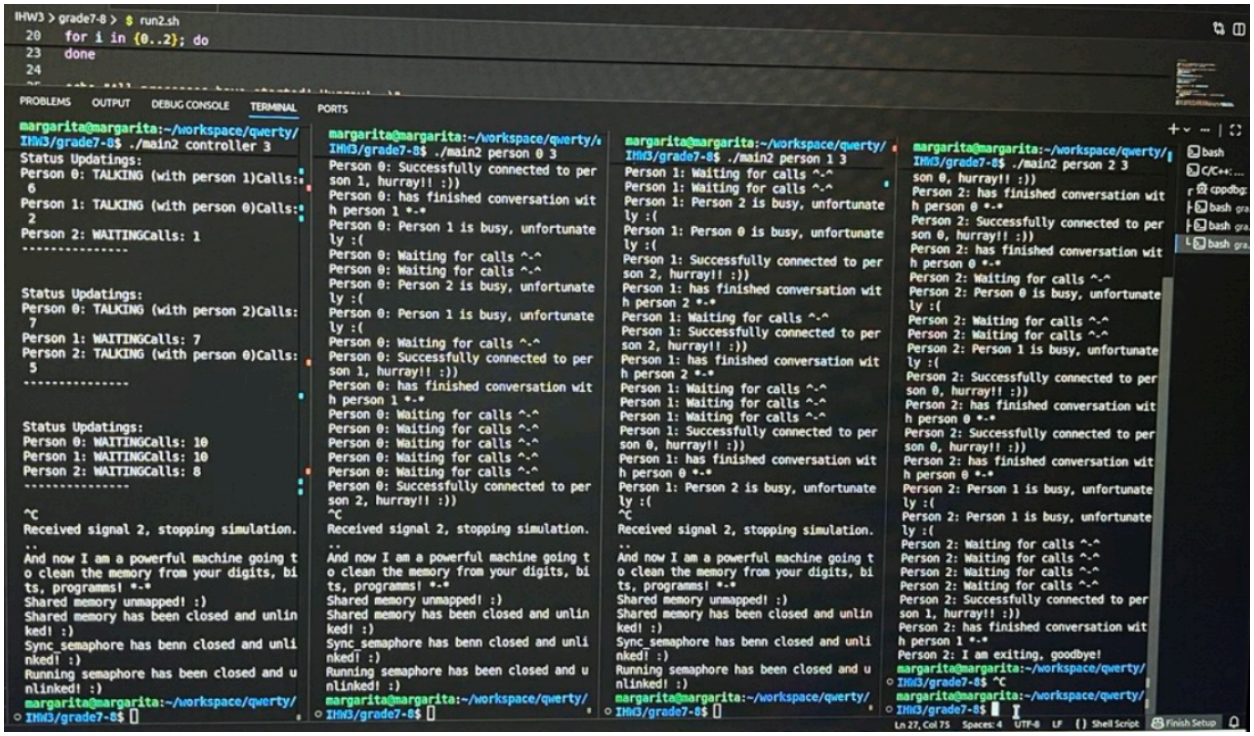
./main3 person 2 3

б) Запустить скрипт: изначально нужно сделать скрипт исполняемым посредством команды chmod +x run2.sh, то есть даем права на исполнение, далее запускаем ./run2.sh.

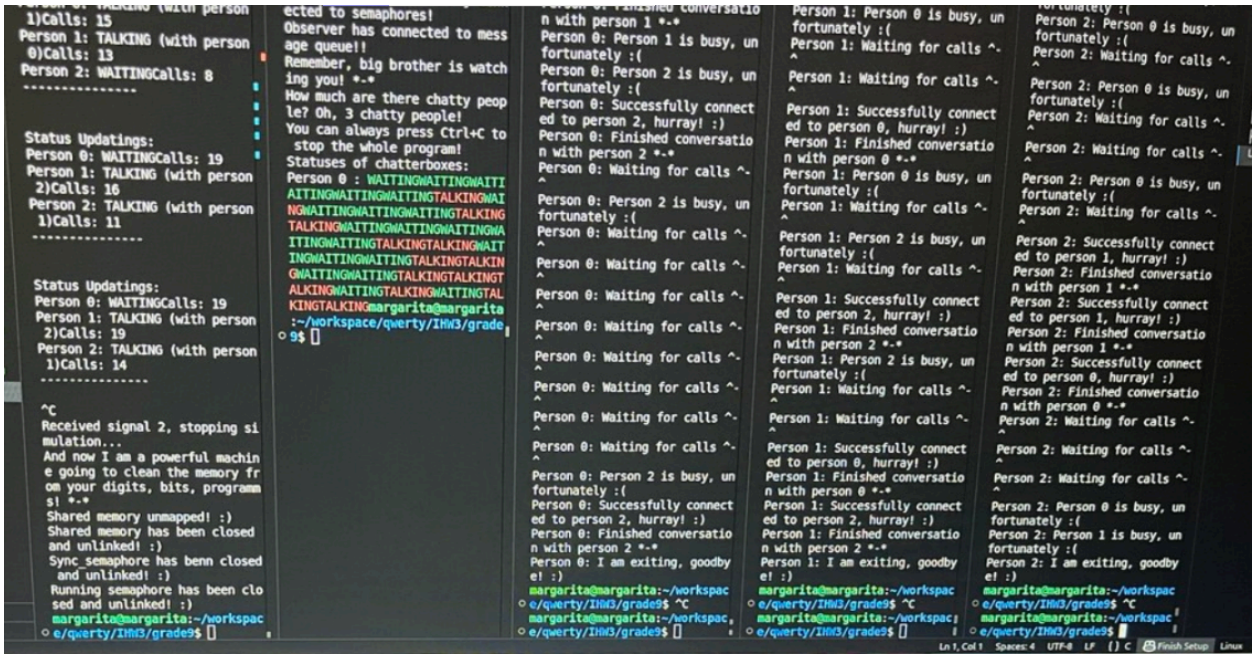
Демонстрация работы программы:

1) 

2)



3)



P.S. цветом выделены статусы болтунов, которые видит observer, с ними все хорошо, это никакие не предупреждения об ошибках https://www.geeksforgeeks.org/linux-unix/how-to-change-the-output-color-of-echo-in-linux/ - идею взяла отсюда.

Программа 1. Review.

Первая программа использует неименованные семафоры и реализует модель "родитель-детки", где родительский процесс порождает дочерние. Разделяемая память хранит статусы всех болтунов, управляет всеми процессами, а неименованные семафоры нужны для синхронизации доступа к этой разделяемой памяти(не зря ведь она и называется разделяемой :)) То есть все дочерние процессы запускаются из одного родительского, увидеть это можно в main'е - там мы форкуем процесс вот так:

```
pid_t pids[NUMBER_OF_CHATTIES];

        //create filia processes
        printf("Creating %d chatty people processes! ^-^\n", num_people);
        for(int i = 0; i < num_people; i++) {
            pid_t pid = fork();

            if (pid == 0) {
                chatty_person_process(i, num_people);
                exit(0);
            } else if (pid > 0) {
                pids[i] = pid;
            } else {
                perror("fork failed");
                exit(1);
            }
        }
```

Ну и все хорошо, вроде, завершаем нажатием Ctrl+C

Программа 2. Review.

Вторая программа использует именованные семафоры и состоит из независимых процессов, запускаемых отдельно в разных консолях. Например, sem_open() вместо sem_init(). Нет общего родительского процесса, каждый процесс запускается отдельной командой. Процессы находят друг друга через системные ресурсы. То есть, как я писала выше, мы запускаем несколько терминалов для контроллера и для болтунов.
Данный блок выполняется только в режиме контроллера.

```
char *mode = argv[1]; //the mode of working: controller or chatterbox

    if (strcmp(mode, "controller") == 0) {
```

```
        int numberOfPeople = atoi(argv[2]);
        if (numberOfPeople < 2 || numberOfPeople > NUMBER_OF_CHATTIES) {
            fprintf(stderr, "Number of people must be between 2 and %d\n",
NUMBER_OF_CHATTIES);
            exit(1);
        }
```

А вот этот только в режиме болтуна.

```
else if (strcmp(mode, "person") == 0) { //if the mode of a chatterbox, he
is in simulation
        if (argc != 4) {
            fprintf(stderr, "Usage: %s person <person_id>
<number_of_people>\n", argv[0]);
            exit(1);
        }
```

Программа 3. Review.

В третьей программе же добавляются ~~танцы с бубнами~~ процесс-наблюдатель, который обеспечивает централизованное отображение информации через очередь сообщений. Добавляется очередь сообщений. Так, болтуны отправляют сообщения о своих действиях, а наблюдатель получает и красиво отображает информацию(привет сайту geeksforgeeks и его инструкции по работе с цветным выводом). Типы сообщений: статус, звонок, соединение, конец разговора. Однако единым неизменный условием для всех программ является использование специальных семафорчиков для защиты разделяемой памяти. При изменении состояния болтуна процесс сначала захватывает семафор, затем изменяет данные, после чего освобождает семафор.

```
//func to send msg to person
void send_message(int person_id, const char *text, int mtype, int
target_id, int calls) {
    //here we will open the message queue
    int local_msgq = msgget(MESSAGE_Q_KEY, 0666);
    if (local_msgq != -1) {
        Message msg;
        msg.mtype = mtype;
        msg.person_id = person_id;
        msg.target_id = target_id;
        msg.numberOfCalls = calls;
        strncpy(msg.mtext, text, MAX_MESSAGE_SIZE - 1);
        msg.mtext[MAX_MESSAGE_SIZE - 1] = '\0';
```

```
        //vstante v ochered'
        if (msgsnd(local_msgq, &msg, sizeof(Message) - sizeof(long),
IPC_NOWAIT) == -1) {

        }
    }
}
```