Министерство образования Республики Беларусь
Учреждение Образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра электронных вычислительных машин

Лабораторная работа № 3
«Взаимодействие и синхронизация процессов.»

Проверил: Поденок Л.П.
Выполнил: ст. гр. 350501
Маслаков Н.А.

Минск 2025

1. УСЛОВИЕ ЛАБАРАТОРНОЙ РАБОТЫ

Синхронизация процессов с помощью сигналов и обработка сигналов таймера.

Задание:

Управление дочерними процессами и упорядочение вывода в stdout от них, используя сигналы SIGUSR1 и SIGUSR2.

Действия родительского процесса По нажатию клавиши «+» родительский процесс (P) порождает дочерний процесс (C_k) и сообщает об этом.

По нажатию клавиши «-» P удаляет последний порожденный C_k, сообщает об этом и о количестве оставшихся.

При вводе символа «l» выводится перечень родительских и дочерних процессов.

При вводе символа «k» P удаляет все C_k и сообщает об этом.

По нажатию клавиши «q» P удаляет все C_k, сообщает об этом и завершается.

Действия дочернего процесса

Дочерний процесс во внешнем цикле заводит будильник и входит в вечный цикл, в котором заполняет структуру, содержащую пару переменных типа int, значениями {0, 0} и {1, 1} в режиме чередования. Поскольку заполнение не атомарно, в момент срабатывания будильника в структуре может оказаться любая возможная комбинация из 0 и 1.

При получении сигнала от будильника проверяет содержимое структуры, собирает статистику и повторяет тело внешнего цикла. Через заданное количество повторений внешнего цикла дочерний процесс выводит свои PPID, PID и 4 числа — количество разных пар, зарегистрированных в момент получения сигнала от будильника.

2. ОПИСАНИЕ АЛГОРИТМОВ И РЕШЕНИЙ

Родительский процесс управляет созданием, удалением и взаимодействием с дочерними процессами. Он обрабатывает команды пользователя, такие как создание нового дочернего процесса, удаление процессов, остановка и возобновление их работы. Также он отслеживает состояние дочерних процессов и их вывод.

Дочерний процесс выполняет цикл, в котором обновляет статистику (количество пар 00, 01, 10, 11). Он отправляет статистику родительскому процессу через сигналы и ожидает разрешения от родительского процесса для вывода данных.

3. ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА

Родительский процесс.

`void InitSignals()`.

Функция `InitSignals` устанавливает обработку сигналов SIGUSR1, SIGUSR2, SIGCHLD.

`void DeleteLastChild()`.

Функция `DeleteLastChild` используется для удаления последнего созданного процесса.

`void CreateChild()`.

Функция `CreateChild` используется для создания нового дочернего процесса и добавления информации о нем в массив `child_processes`.

`void HandleSignal(int signo, signinfo_t *info, void *context)`.

Функция `HandleSignal` используется для Обработки сигналов от дочерних процессов.

Принимаемые параметры:

1) `int signo`. Номер получаемого сигнала.

2) `siginfo_t *info`. Информация о сигнале.

3) `void *context`. Не используется.

`void ListChild()`.

Функция `ListChild` используется для вывода списка всех дочерних процессов и их информации.

`void DeleteAllChild()`.

Функция `DeleteAllChild` используется для удаления всех дочерних процессов.

`void CleanExit()`.

Функция `CleanExit` используется для удаления всех дочерних процессов и завершения родительского.

`void WaitForChildren()`.

Функция `WaitForChildren` используется для завершения всех оставшихся процессов с помощью `waitpid()`.

`void PrintMenu()`.

Функция `PrintMenu` используется для вывода пунктах меню.

`void StartChild(int index)`.

Функция `StartChild` используется для разрешения вывода дочернему процессу по заданному индексу.

Принимает параметр `int index` - индекс дочернего процесса.

`void StopChild(int index)`.

Функция `StopChild` используется для запрета вывода дочернему процессу по заданному индексу.

Принимает параметр `int index` - индекс дочернего процесса.

Функция `main` является основной точкой входа в программу, которая инициализирует обработку сигналов, выделяет память для хранения информации о дочерних процессах, выводи меню, а затем, в бесконечном цикле ожидает ввода для выполнения команды.

Дочерний процесс.

`void InitSignalsHandling()`.

Функция `InitSignalsHandling` используется для настройки обработки сигналов для дочернего процесса.

`void UsrSignalHandling(int signo)`.

Функция `UserSignalHandling` используется для обработки пользовательских сигналов.

Принимает параметр `int signo` – номер полученного сигнала.

`void AlrSignalHandler(int signo)`.

Функция `AlrSignalHandler` используется для обработки сигнала будильника, обновляя статистику на основе текущего состояния.

Принимает параметр `int signo` – номер полученного сигнала.

`void UpdateStat()`.

Функция `UpdateStat` используется для циклического обновления полей структуры `occurance` в фиксированной последовательности.

Функция `main` является основной точкой входа в программу, которая инициализирует обработку сигналов, устанавливает будильник со случайным интервалом и в бесконечном цикле обновляет статистику.

4. ПОРЯДОК СБОРКИ И ЗАПУСКА

1) Перейти в каталог проекта.

```
$ cd 'Маслаков Н.А./lab03'
```

2) Собрать проект с помощью `make`. по умолчанию сборка происходит в режиме отладки.

```
$ make
```

3) Запустить программу.

```
$ ./build/debug/parent
```

5. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

```
~/lab03$ ./build/debug/parent
Options:
+ - Create new child
- - Delete last child
l - List all children
k - Delete all children
s<num> - Stop child at index <num>
g<num> - Start child at index <num>
q - Quit
m - Show this menu
> +
Created child C_00 with PID 5701
> g0
Started child C_00 with PID 5701
> Parent: Received SIGUSR1 from child 5701
> ----------------------------------------
ppid -  5700    pid  -  5701    00   -     2; 01   -     1; 10  -
1; 11   -     1
Parent: Child 5701 has finished output
> Parent: Received SIGUSR1 from child 5701
> ----------------------------------------
ppid -  5700    pid  -  5701    00   -     3; 01   -     2; 10  -
2; 11   -     3
Parent: Child 5701 has finished output
> Parent: Received SIGUSR1 from child 5701
> ----------------------------------------
ppid -  5700    pid  -  5701    00   -     4; 01   -     3; 10  -
3; 11   -     5
Parent: Child 5701 has finished output
> s0
Stopped child C_00 with PID 5701
> +
Created child C_01 with PID 5705
> g1
Started child C_01 with PID 5705
> Parent: Received SIGUSR1 from child 5705
> ----------------------------------------
ppid -  5700    pid  -  5705    00   -     2; 01   -     0; 10  -
1; 11   -     2
Parent: Child 5705 has finished output
> Parent: Received SIGUSR1 from child 5705
> ----------------------------------------
ppid -  5700    pid  -  5705    00   -     3; 01   -     1; 10  -
2; 11   -     4
Parent: Child 5705 has finished output
> Parent: Received SIGUSR1 from child 5705
> ----------------------------------------
ppid -  5700    pid  -  5705    00   -     4; 01   -     2; 10  -
3; 11   -     6
Parent: Child 5705 has finished output
> s1
```

```
Stopped child C_01 with PID 5705
> g0
Started child C_00 with PID 5701
> Parent: Received SIGUSR1 from child 5701
> ----------------------------------------
ppid -  5700    pid  -  5701    00   -    12; 01   -    11; 10  -
11; 11   -    19
Parent: Child 5701 has finished output
> l
Parent PID: 5700
Child C_00 with PID 5701 is stopped
Child C_01 with PID 5705 is stopped
> -
Deleted child C_01 with PID 5705
> > l
Parent PID: 5700
Child C_00 with PID 5701 is stopped
> q
Deleted child C_00 with PID 5701
All children deleted
Exiting...
```