Лабораторна робота № 2.1

КЕРУВАННЯ СИСТЕМНИМИ РЕСУРСАМИ ЗАСОБАМИ SHELL-ІНТЕРПРЕТАТОРА

Мета – ознайомитися з основами програмування на рівні командної мови Shell шляхом написання Shell-програм для роботи з файловою системою.

Зміст роботи

1. Вивчити програмні засоби мови Shell (структура команди, групування команд, перенаправлення введення-виведення, конвеєр команд, Shell-змінні, макропідстановка результатів в Shell-командах, програмні конструкції).
2. Ознайомитися із завданням.
3. Для вказаного варіанта скласти Shell-програму, що виконує необхідні дії у файловій системі.
4. Налагодити і відтестувати складену Shell-програму.
5. Захистити роботу, відповівши на контрольні запитання.

Короткі теоретичні відомості

Зазвичай в ОС UNIX доступні декілька інтерпретаторів. Найбільш поширені Bourne-shell (чи просто Shell), C-shell, Korn-shell. В ідейному плані усі ці інтерпретатори близькі і надалі йтиметься про стандартний Shell (/bin/sh).

Працюючи на командній мові, користувач може вводити змінні, привласнювати їм значення, виконувати прості команди, будувати складені команди, управляти потоком виконання команд, об'єднувати послідовність команд в процедури (командні файли). На рівні командної мови доступні такі властивості системи, як з'єднання процесів через програмний канал, напрям стандартного введення/виведення в конкретні файли, синхронне і асинхронне виконання команд.

Якщо вказаний інтерпретатору файл є текстовим і містить команди командної мови (командний файл) і при цьому має дозвіл на виконання (помічений "x"), Shell-інтерпретатор інтерпретує і виконує команди цього файлу. Інший спосіб виклику командного файлу – використання команди sh (виклик інтерпретатора), в якому першим аргументом вказується ім'я командного файлу.

Коротко перерахуємо засоби групування команд і перенаправлення введення/виведення:

- **cmd1 arg ...; cmd2 arg ...; ... cmdN arg ...** – послідовне виконання команд;
- **cmd1 arg ... & cmd2 arg ... & ... cmdN arg ...** – асинхронне виконання команд;
- **cmd1 arg ... && cmd2 arg ...** – залежність наступної команди від попередньої таким чином, що наступна команда виконується, якщо попередня видала нульове значення;
- **cmd1 arg ... || cmd2 arg ...** – залежність наступної команди від попередньої таким чином, що наступна команда виконується, якщо попередня видала ненульове значення;
- **cmd > file** – стандартне виведення направлене в файл file;

- **cmd >> file** – стандартне виведення направлене в кінець файлу file;
- **cmd < file** – стандартне введення виконується з файлу file;
- **cmd1 | cmd2** – конвеєр команд, в якому стандартне виведення команди cmd1 направлене на стандартне введення команди cmd2.

Shell-змінні можуть зберігати рядки тексту. Правила формування їх імен аналогічні правилам задання імен змінним в звичайних мовах програмування. При необхідності присвоїти Shell-змінній значення, що містить пропуски й інші спеціальні знаки, воно береться в лапки. При використанні Shell-змінної у вираженні її імені повинен передувати знак $. В послідовності символів ті з них, які складають ім'я, мають бути виділені в { } або " ". Крім того, інтерпретатор Shell автоматично привласнює значення п'яти своїм змінним:

- **$?** – значення, яке повертається останньою виконуваною командою;
- **$$** – ідентифікаційний номер процесу Shell;
- **$!** – ідентифікаційний номер фонового процесу, який запускався інтерпретатором Shell останнім;
- **$#** – кількість аргументів, переданих в Shell;
- **$-** – прапорці, передані в Shell.

Для відміни спеціальних символів (**$**, **|**, пробіл і т.д.) в Shell-програмах існують такі правила:

- якщо символу передує обернена коса риска, то його спеціальний символ відміняється;
- відміняється спеціальний сенс усіх символів, що увійшли до послідовності, яка розміщена в апострофах.

Під час виклику Shell-програм їм можуть передаватися параметри. Відповідні аргументи в Shell-програмах ідентифікуються **$1, $2, $3** і так далі. Крім того, змінна **$0** відповідає імені виконуваної Shell-програми, а змінна **$#** – кількості аргументів в команді.

Shell-інтерпретатор дає можливість виконувати підстановку результатів виконання команд в Shell-програмах. Якщо команда береться в одинарні зворотні лапки, то інтерпретатор Shell виконує цю команду і підставляє замість неї отриманий результат.

Найбільш важливі команди для складання Shell-програм:

- команда **echo** виводить у вихідний потік значення своїх аргументів;
- команда **expr** виконує арифметичні дії над своїми аргументами;
- команда **eval** забезпечує додатковий рівень підстановки своїх аргументів, а потім їх виконання;
- команда **test** з відповідними ключами перевіряє необхідні умови;
- команда **sleep** слугує для реалізації затримки.

Програмні конструкції Shell-програм:

Оператори циклу

Цикл for

```
for змінна in список_слів
do
    команда
done
```

Ця команда послідовно привласнює всі значення із списку слів у змінну і виконує команди в тілі циклу.

Цикл while

while команда
do
тіло циклу
done

Цикл працює доти, доки успішно виконується команда біля while.

Цикл until

until команда
do
тіло циклу
done

Цикл працює доти, доки результат виконання команди буде невірним.

Оператори вибору

Умовний оператор if

If команда
then команда_1
else команда_2
fi

Якщо команда виконана успішно, то виконується команда_1, інакше – команда_2.

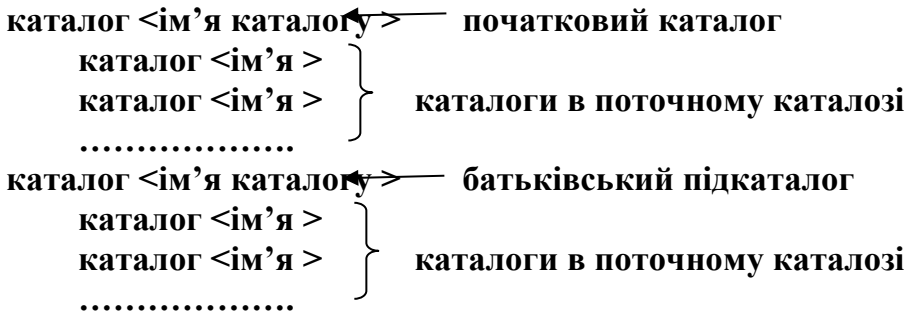
Структура case

case слово **in**
шаблон) команда_1 ;;
шаблон) команда_2 ;;
...
esac

Слово порівнюється зі всіма шаблонами та виконує команду при першому збіжному шаблоні.

ВАРІАНТИ ЗАВДАННЯ

1. Shell-програма виводить з каталогу імена тих каталогів, які в собі містять каталоги. Ім'я каталогу задане параметром Shell-програми.
2. Shell-програма переглядає каталог, ім'я якого вказане параметром Shell-програми і виводить імена каталогів, що зустрілися. Потім здійснює перехід в батьківський каталог, який стає поточним, і повторюються вказані дії доти, доки поточним каталогом не стане кореневий каталог. Форма виведення результату:



і т.д.

3. Shell-програма підраховує кількість і виводить перелік каталогів в хронологічному порядку (за датою створення) в піддереві, починаючи з каталогу, ім'я якого задане параметром Shell -програми. Форма виведення результату:

каталог <ім'я каталогу> початковий каталог
каталог <ім'я >
каталог <ім'я > } каталоги в поточному каталозі
…….

каталог <ім'я каталогу> підкаталог
каталог <ім'я >
каталог <ім'я > } каталоги в поточному каталозі
…….
і т.д.

4. Shell-програма об'єднує усі тимчасові файли з вказаним суфіксом (наприклад, .tmp) в піддереві, починаючи з каталогу, ім'я якого задане параметром Shell-програми. Результат об'єднання поміщається або у вказаний Shell-програмою файл, або виводиться на екран у формі:

<ім'я каталогу >:<ім'я файлу> початковий каталог
[вміст файлу]
End of file
<ім'я файлу>
[вміст файлу]
End of file
…….
<ім'я каталогу >:<ім'я файлу> підкаталог
[вміст файлу]
End of file
<ім'я файлу>
[вміст файлу]
End of file
…….

і т.д.

5. Shell-програма періодично з деяким інтервалом видаляє усі тимчасові файли з вказаним суфіксом (наприклад, .tmp) в піддереві, починаючи з каталогу, ім'я якого задане параметром Shell-програми і виводить при цьому список файлів, що залишилися, у формі:

каталог <ім'я каталогу> початковий каталог
<ім'я файлу > <довжина>
<ім'я файлу > <довжина> } файли каталогу

…….
каталог <ім’я каталогу > — підкаталог
<ім’я файлу > <довжина>
<ім’я файлу > <довжина> файли каталогу
…….
і т.д.

6. Shell-програма виводить вміст каталогу, ім'я якого вказане параметром Shell-програми. При виведенні спочатку перераховуються імена каталогів, а потім в алфавітному порядку імена файлів з вказівкою їх довжини, дати створення і кількості посилань на них.
7. Shell-програма підраховує кількість і виводить список усіх файлів (без каталогів) в порядку зменшення їх довжини в піддереві, починаючи з каталогу, ім'я якого задане параметром Shell-програми. Форма виведення результату:

каталог <ім’я каталогу > — початковий каталог
<ім’я файлу > <довжина>
<ім’я файлу > <довжина> файли каталогу
…….
каталог <ім’я каталогу > — підкаталог
<ім’я файлу > <довжина>
<ім’я файлу > <довжина> файли каталогу
…….
і т.д.

8. Shell-програма переглядає каталог, ім'я якого вказане параметром Shell-програми і виводить імена файлів, що зустрілися. Потім здійснює перехід в батьківський каталог, який стає поточним і повторюються вказані дії доти, доки поточним каталогом не стане кореневий каталог. Форма виведення результату:

каталог <ім’я каталогу > — початковий каталог
каталог <ім’я >
каталог <ім’я > — каталоги в поточному каталозі
…….
каталог <ім’я каталогу > — батьківський підкаталог
каталог <ім’я >
каталог <ім’я > — каталоги в поточному каталозі
…….
і т.д.

9. Shell-програма підраховує кількість і виводить список усіх файлів (без каталогів) в алфавітному порядку в піддереві, починаючи з каталогу, ім'я якого задане параметром Shell-програми. Форма виведення результату:

каталог <ім’я каталогу > — початковий каталог
<ім’я файлу > <довжина>
<ім’я файлу > <довжина> файли каталогу
…….

каталог <ім'я каталогу> підкаталог
<ім'я файлу > <довжина>
<ім'я файлу > <довжина> файли каталогу
…….
і т.д.

10. Shell-програма виводить з каталогу імена тих каталогів, які в собі не містять каталогів. Ім'я каталогу задане параметром Shell-програми.

Контрольні запитання та завдання

1. Що таке внутрішні і зовнішні команди Shell-інтерпретатора? Наведіть приклади внутрішніх команд.
2. Які існують засоби групування команд? Наведіть приклади.
3. Як реалізується перенаправлення введення-виведення?
4. В чому суть конвеєра команд? Наведіть приклади використання.
5. Як засобами Shell виконати арифметичні дії над Shell-змінною?
6. Які правила генерації імен файлів?
7. Як відбувається підстановка результатів виконання команд?
8. Як інтерпретувати рядок **cmd1 & cmd2 &** ?
9. Як інтерпретувати рядок **cmd1 && cmd2 &** ?
10. Як інтерпретувати рядок **cmd1 || cmd2 &** ?
11. В якому режимі виконується інтерпретатор команд Shell?
12. Ким і в якому режимі здійснюється читання потоку символів з терміналу інтерпретатором Shell?

Лабораторна робота № 2.2

Тема: „Використання компілятора С у середовищі UNIX”

Мета роботи – отримати навички з використання компілятора С у середовищі UNIX.

Теоретична інформація

Текст програми С створюється будь-яким текстовим редактором системи і зберегти в текстовий файл з розширенням .c (наприклад prog.c).

Виклик компілятору здійснюється з командного рядку командою cc. (Наприклад prog.c). Відтрансльований файл зберігається в файл a.out. Файл a.out можна перейменувати та зробити таким, що виконується.

Завдання

1. Написати програму знаходження максимального, мінімального і середньоарифметичного значень ряду з 8 чисел.
2. Протабулювати функцію y= 1/(1-cos(x)) в заданих межах із заданим кроком.
3. Написати програму переводу строки у шістнадцятирічний дамп.
4. Написать програму заміни у рядку одного символу на інший.
5. Написати програму перетворення строки в її дзеркальне відображення.
6. Написати програму, яка копіює задані текстові файли, виключаючи символи з кодами 128-255.
7. Написати програму, яка шифрує зміст заданого файлу методом Цезаря (ключ задавати через командний рядок).
8. Написати програму, яка шукає у заданому файлі слово.
9. Написати програму, яка виділяє з текстового файлу цифрові символи та зберігає їх у іншому файлі (ctype.h, stdlib.h).
10. Написати програму, яка прибирає зайві символи (!! → !) та пробіли в тексті.
11. Написати програму, яка записує зміст текстового файлу в зворотному порядку.
12. Написати програму, яка записує введене слово в задану позицію текстового файлу.
13. Написати програму, яка підраховує функцію (наприклад, sin(x)+x-1) та записує результат обчислення в окремий файл (крок та відрізок вводити з клавіатури). (math.h)
14. Написати програму, яка переводить символи алфавіту текстового файлу в їх еквівалентний числовий формат (код символу) та записує в окремий файл.
15. Написати програму, яка перевіряє ідентичність двох текстових файлів, результат виводить у вигляді: дата перевірки (ctime.h), назва файлів, результат перевірки та відсоток ідентичності файлів.

Лабораторна робота № 2.3

ФАЙЛОВА СИСТЕМА ОС UNIX

Мета – ознайомитися з файловою системою ОС UNIX, механізмами її функціонування, основними елементами файлової системи: суперблок, описувачі файлів, типи файлів, список вільних описувачів файлів, список вільних блоків.

Зміст роботи

1. Ознайомитися з файловою системою ОС UNIX і програмними засобами роботи з нею.
2. Ознайомитися з завданням.
3. Навчитися складати програми на мові Сі, що реалізує необхідні дії.
4. Налагодити і відтестувати складенні програми, використовуючи інструментарій ОС UNIX.
5. Захистити роботу, відповівши на контрольні запитання.

Короткі теоретичні відомості

Інтерфейс між користувацькою програмою і зовнішнім пристроєм (чи між двома користувацькими програмами) в ОС UNIX здійснюється у рамках єдиної структури даних, яка називається файлом ОС UNIX.

Всякий файл ОС UNIX відповідно до його типу може бути віднесений до однієї з таких чотирьох груп: звичайні файли, каталоги, спеціальні файли, канали.

Звичайний файл є сукупністю блоків диска, що входять до складу файлової системи ОС UNIX. У вказаних блоках може бути довільна інформація.

Каталоги є файлами особливого типу, що відрізняються від звичайних, передусім, тим, що здійснити запис в них може тільки ядро ОС UNIX, тоді як доступ до читання може отримати будь-який призначений для користувача процес, що має відповідні повноваження. Кожен елемент каталогу складається з двох полів: поля імені файлу і поля, що містить покажчик на описувач файлу, де зберігається уся інформація про файл: дата створення, розмір, код захисту, ім'я власника і т.д. У будь-якому каталозі міститься, принаймні, два елементи, що містять в полі імені файлу імена "." і "..". Елемент каталогу, що містить в полі імені файлу контекст ".", в полі посилання містить посилання на описувач файлу, який описує цей каталог. Елемент каталогу, що містить в полі імені файлу контекст "..", в полі посилання містить посилання на описувач файлу, в якому зберігається інформація про батьківський каталог поточного каталогу.

Спеціальні файли – це деякі файли, кожному з яких ставиться у відповідність свій зовнішній пристрій, який підтримує ОС UNIX і що має спеціальну структуру. Його не можна використовувати для зберігання даних, як звичайний файл або каталог. В той же час над спеціальним файлом можна проводити ті самі операції, що і над звичайним файлом: відкривати, вводити і виводити інформацію і так далі. Результат застосування будь-якої з цих операцій залежить від того, якому

конкретному пристрою відповідає оброблюваний спеціальний файл, проте, у будь-якому випадку буде здійснена відповідна операція введення-виведення на зовнішній пристрій, якому відповідає вибраний спеціальний файл.

Четвертий вид файлів – канали.

Системні функції ОС UNIX для роботи з файловою системой

Повертають дескриптор файлу	**open, creat, dup, pipe, close**
Перетворюють ім’я в описувач	**open, creat, chdir, chmod, stat, mkfifo, mound, mknod, link, unmount, unlink, chown**
Назначають inode	**creat, link, unlink, mknod**
Працюють з атрибутами	**chown, chmod, stat**
Введення/виведення із файлу	**read, write, lseek**
Працюють із структурою файлової системи	**mount, unmount**
Керують деревами	**chmod, chown**

Зупинимось на тих з них, які необхідні для виконання роботи. Для отримання інформації про тип файлу необхідно скористатися системними викликами **stat()** (**fstat()**). Формат системних викликів **stat()** (**fstat()**):

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(const char *name, struct stat *stbuf);
int fstat(int fd, struct stat *stbuf);
```

Обидва системні виклики розміщують інформацію про файл (у першому випадку специфікованому ім'ям **name**, а в другому – дескриптором файлу **fd**) в структурну змінну, на яку вказує **stbuf**. Функція, яка робить виклик, повинна потурбуватися про резервування місця для інформації, що повертається; у разі успіху повертається **0**, інакше – **1** і код помилки в **errno**. Опис структури **stat** міститься у файлі **sys/stat.h**. З невеликими модифікаціями вона має вигляд:

```
struct stat
{
 dev_t st_dev; /* device file */
 ino_t st_ino; /* file serial inode */
ushort st_mode; /* file mode */
short st_nlink; /* number of links */
ushort st_uid; /* user ID */
```

```
ushort st_gid; /* group ID */
dev_t st_rdev; /* device ident */
off_t st_size; /* size of file */
time_t st_atime; /* last access time */
time_t st_mtime; /* last modify time */
time_t st_ctime; /* last status change */
};
```

Поле **st_mode** містить прапорці, які описують файл. Прапорці несуть таку інформацію:

S_IFMT 0170000 – тип файлу;
S_IFDIR 0040000 – каталог**;**
S_IFCHR 0020000 – байт-орієнтований спеціальний файл;
S_IFBLK 0060000 – блок-орієнтований спеціальний файл**;**
S_IFREG 0100000 – звичайний файл;
S_IFFIFO 0010000 – дисципліна FIFO;
S_ISUID 04000 – ідентифікатор власника;
S_ISGID 02000 – ідентифікатор групи;
S_ISVTX 01000 – зберегти свопіюємий текст**;**
S_ISREAD 00400 – власнику дозволено читання;
S_IWRITE 00200 – власнику дозволено запис;
S_IEXEC 00100 – власнику дозволено виконання;

Символьні константи, чотири перших символи, які співпадають з контекстом **S_IF**, можуть бути використані для визначення типу файлу.

Більшість системних викликів, які працюють з каталогами, оперують структурою **dirent**, яка визначена в заголовному файлі **<dirent.h>**.

```
struct dirent
{
ino_t d_ino; /* номер индексного дескриптора */
char d_name[DIRSIZ]; /* имя файла */
}
```

Створення і видалення каталогу виконується системними викликами **mkdir()** і **rmdir()**. При створенні каталогу за допомогою системного виклику **mkdir()** в нього поміщаються два посилання ("**.**" і "**..**").

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int mkdir (char *pathname, mode_t mode);
int rmdir (char *pathname);
```

Відкриття і закриття каталогу виконується системними викликами **opendir()** і **closedir()**. При успішному відкритті каталогу системний виклик повертає покажчик на змінну типу **DIR**, що є дескриптором каталогу, визначену у файлі **dirent.h** і використовувану при читанні і записі в каталог. При невдалому виклику повертається значення **NULL**.

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir (char *dirname);
int closedir (DIR *dirptr); /* dirptr - дескриптор каталога */
```

Для зміни каталогу служить системний виклик **chdir()**:

```
#include <unistd.h> int chdir (char *pathname);
```

Читання записів каталогу виконується системним викликом **readdir().** Системний виклик **readdir()** за номером дескриптора каталогу повертає черговий запис з каталогу в структуру dirent, або нульовий покажчик при досягненні кінця каталогу. При успішному читанні, покажчик каталогу переміщається до наступного запису. Додатковий системний виклик **rewinddir()** переводить покажчик каталогу до першого запису каталогу.

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir (DIR *dirptr);
void rewinddir (DIR *dirptr);
```

ВАРІАНТИ ЗАВДАННЯ

1. Розробити програму, яка здійснює перегляд поточного каталогу і виводить на екран його вміст групами в порядку зростання кількості посилань на файли (у тому числі імена каталогів). Група є об'єднанням файлів з однаковою кількістю посилань на них.
2. Розробити програму, яка переглядає поточний каталог і виводить на екран імена усіх файлів, що зустрілися в ньому, із заданим розширенням. Потім здійснюється перехід в батьківський каталог, який потім стає поточним, і вказані вище дії повторюються доти, доки поточним каталогом не стане кореневий каталог.
3. Розробити програму, яка переглядає поточний каталог і виводить на екран імена усіх звичайних файлів, що зустрілися в ньому. Потім здійснюється перехід в батьківський каталог, який потім стає поточним, і вказані вище дії повторюються доти, доки поточним каталогом не стане кореневий каталог.

4. Розробити програму, яка виводить на екран імена тих каталогів, які знаходяться в поточному каталозі і не містять в собі підкаталогів.
5. Розробити програму, яка виводить на екран імена тих каталогів, які знаходяться в поточному каталозі і містять в собі підкаталоги.
6. Розробити програму, яка виводить на екран вміст поточного каталогу, впорядкований за часом створення файлів. При цьому імена каталогів повинні виводитися останніми.
7. Розробити програму, яка виводить на екран вміст поточного каталогу в порядку зростання розмірів файлів. При цьому імена каталогів повинні виводитися першими.
8. Розробити програму, яка виводить на екран вміст поточного каталогу в алфавітному порядку. Каталоги не виводити.
9. Розробити програму, яка переглядає поточний каталог і виводить на екран імена усіх каталогів, що зустрілися в ньому. Потім здійснюється перехід в батьківський каталог, який потім стає поточним, і вказані вище дії повторюються доти, доки поточним каталогом не стане кореневий каталог.
10. Розробити програму, яка здійснює перегляд поточного каталогу і виводить на екран імена каталогів, що знаходяться в ньому, упорядкувавши їх за кількістю файлів і каталогів, які містяться в каталозі, що відображується. Для кожного такого каталогу вказується кількість файлів і каталогів, що містяться в ньому.

Контрольні запитання та завдання

1. Що являє собою суперблок?
2. Що таке список вільних блоків?
3. Що являє собою список описувачів файлів?
4. Як проводиться виділення вільних блоків під файл?
5. Як проводиться звільнення блоків даних, зайнятих під файл?
6. Яким чином здійснюється монтування дискових пристроїв?
7. Яке призначення елементів структури stat?
8. Яким чином здійснюється захист файлів в ОС UNIX?
9. Які права доступу до файлу, при яких власник може виконувати усі операції **(r, w, x)**, а інші користувачі - тільки читати?
10. Що виконує системний виклик **lseek(fd, (off_t)0, SEEK_END)**?

Лабораторна робота № 2.4

МОДЕЛЮВАННЯ РОБОТИ ІНТЕРПРЕТАТОР

Мета роботи: Практичне освоєння засобів управління ресурсами ОС UNIX на основі розробки програми, що моделює роботу інтерпретатора в плані створення процесів, що реалізують команди в командному рядку, їх синхронізації і взаємодії за даними.

Зміст роботи

1. Вивчити програмні засоби спадкування дескрипторів файлів (системні виклики **dup()**, **fcntl()**).

2. Ознайомитися із завданням до лабораторної роботи.

3. Вибрати набір системних викликів, які забезпечують вирішення завдання.

4. Навчитися складати програми мовою Сі, що реалізовує необхідні дії.

5. Налагодити і відтестувати складені програми, використовуючи інструментарій ОС UNIX.

6. Захистити лабораторну роботу, відповівши на контрольні питання.

Методичні вказівки до виконання лабораторних робіт

При виконанні операції перенаправлення вводу-виводу важливим моментом є успадкування користувальницьких дескрипторів, здійснюване за допомогою системних викликів **dup()** і **fcntl()**.

Системний виклик dup () обробляє свій єдиний параметр як користувальницький дескриптор відкритого файлу і повертає ціле число, яке може бути використане як ще один користувальницький дескриптор того ж файлу. За допомогою копії користувацького дескриптора файлу до нього може бути здійснений доступ того ж типу і з використанням того ж значення покажчика запису-читання, що і з допомогою оригінального користувальницького дескриптора файлу.

Системний виклик fcntl(), який має формат

int fcntl(int fd, char command, int argument);

виконує дії з розділення користувальницьких дескрипторів залежно від п'яти значень аргументу command, специфіковані у файлі **fcntl.h**. Наприклад, при значенні другого аргументу, рівного **F_DUPFD**, системний виклик **fcntl ()** повертає перший вільний дескриптор файлу, значення якого не менше значення аргументу **argument**. Цей користувальницький дескриптор файлу повинен бути копією користувальницького дескриптора файлу, заданого аргументом **fd**.

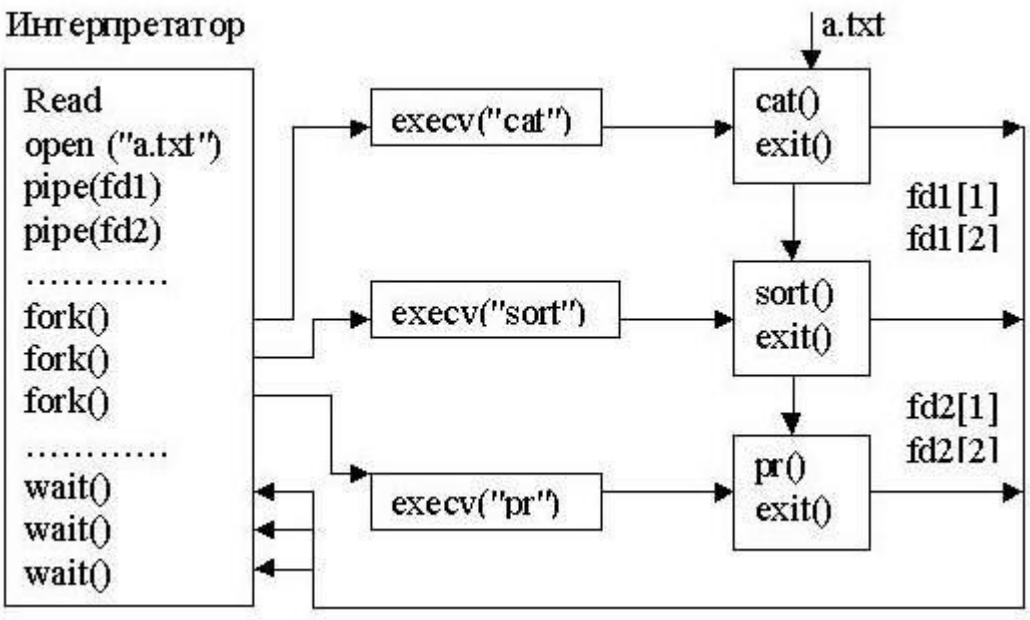
За допомогою системних викликів **dup ()** і **fcntl ()** для користувача програми, а також і інтерпретатор команд Shell реалізують канали і перепризначення стандартного введення і стандартного виводу на файл. Нехай, наприклад, деяка програма **prog** читає дані зі стандартного вхідного потоку і виводить результати в стандартний вихідний потік. Для того, щоб та ж програма читала дані з файлу **aa.txt**, а здійснювало виведення в файл **bb.txt**, необхідно виконати:

#include <fcntl.h> ..... int fd, fd2; fd = open("aa.txt", O_RDONLY); close(0); fcntl(fd,F_DUPFD,0); fd = open("bb.txt", O_WRONLY | O_CREAT); close(1); fcntl(fd2, F_DUPFD, 1); execlp("prog", "prog", 0);

Інтерпретатор Shell являє собою звичайну, з точки зору користувача, програму, яка в ході свого функціонування створює процеси, що реалізують прості команди командного мови, виконує перенаправлення вводу-виводу, будує програмні канали між командами і т.д. Наприклад, схему обробки командного рядка

cat < a.txt | sort | pr

інтерпретатором команд, опускаючи деталі, пов'язані зі спадкуванням дескрипторів файлів, можна представити у вигляді:



Варіанти завдань

Скласти програму, що моделює роботу Shell-інтерпретатора при обробці командного рядка. При реалізації програми шляхом видачі повідомлень інформувати про всі етапи її роботи (створений процес, виконання команди закінчено і т.д.).

1. (cc pr1.c & cc pr2.c) && cat pr1.c pr2.c > prall.c.

2.wc -c < a.txt & wc -c < b.txt & cat a.txt b.txt | wc -c > c.txt.

3.who | wc -l & ps | wc -l.

4.tr -d "[p-z]" < a.txt | wc -c & wc -c < a.txt.

5.ls -la > a.txt & ps > b.txt; cat a.txt b.txt | sort.

6.ls -lisa | sort | wc -l > a.txt.

7.cat a.txt b.txt c.txt | tr -d "[a-i]" | wc -w.

8.ls -al | wc -l && cat a.txt b.txt > c.txt.

9.tr -d "[0-9]" < a.txt | sort | uniq > b.txt.

10. ls -al | grep "April" | wc -l > a.txt.

Лабораторна робота № 2.5

ПОРОДЖЕННЯ НОВОГО ПРОЦЕСУ І РОБОТА З НИМ. ЗАПУСК ПРОГРАМИ В РАМКАХ ПОРОДЖЕНОГО ПРОЦЕСУ. СИГНАЛИ І КАНАЛИ В ОС UNIX

Мета роботи

Вивчити програмні засоби створення процесів, отримати навички управління і синхронізації процесів, а також найпростіші способи обміну даними між процесами. Ознайомитися із засобами динамічного запуску програм в рамках породженого процесу, вивчити механізм сигналів ОС UNIX, що дозволяє процесам реагувати на різні події, і канали, як один із засобів обміну інформацією між процесами.

Зміст роботи

1. Вивчити правила використання системних викликів **`fork ()`, `wait ()`, `exit ()`.**
2. Ознайомитися з системними викликами **`getpid ()`, `getppid ()`, `se tpgrp ()`, `getpgrp ()`.**
3. Вивчити засоби динамічного запуску програм в ОС UNIX (системні виклики **`execl ()`, `execv ()`,** ...).
4. Вивчити засоби роботи з сигналами і каналами в ОС UNIX.
5. Ознайомитися із завданням до лабораторної роботи.
6. Скласти програми на мові Сі, що реалізують завдання.
7. Налагодити і від тестувати складену програму, використовуючи інструментарій ОС UNIX.
8. Захистити лабораторну роботу, відповівши на контрольні питання.

Методичні вказівки до лабораторної роботи

Для породження нового процесу (процес-нащадок) використовується системний виклик **`fork ()`.** Формат виклику:

```
int fork ();
```

Породжений таким чином процес являє собою точну копію свого процесу-предка. Єдина відмінність між ними полягає в тому, що процес-нащадок в якості значення, що повертається системного виклику **`fork ()`** отримує 0, а процес-предок - ідентифікатор процесу-нащадка. Крім того, процес-нащадок успадковує і весь контекст програмного середовища, включаючи дескриптори файлів, канали і т.д. Наявність у процесу ідентифікатора дає можливість і ОС UNIX, і будь-якого іншого користувача процесу отримати інформацію про функціонуючих в даний момент процесах.

Очікування завершення процесу-нащадка батьківським процесом виконується за допомогою системного виклику **wait ():**

```
int wait (int * status);
```

У результаті здійснення процесом системного виклику **wait()** функціонування процесу припиняється до моменту завершення породженого ним процесу-нащадка. По завершенні процесу-нащадка процес-предок пробуджується і в якості значення, що повертається системного виклику **wait()** отримує ідентифікатор завершився процесу-нащадка, що дозволяє процесу-предка визначити, який з його процесів-нащадків завершився (якщо він мав більше одного процесу-нащадка) . Аргумент системного виклику **wait()** являє собою вказівник на цілочисельну змінну змінну **status,** яка після завершення виконання цього системного виклику буде містити в старшому байті код завершення процесу-нащадка, встановлений останнім в якості системного виклику **exit (),** а в молодшому - індикатор причини завершення процесу -потомка.

Формат системного виклику **exit (),** призначеного для завершення функціонування процесу:

```
int exit (int status);
```

Аргумент **status** є статусом завершення, який передається батькові процесу, якщо він виконував системний виклик **wait ().**

Для отримання власного ідентифікатора процесу використовується системний виклик **getpid (),** а для отримання ідентифікатора процесу-батька - системний виклик **getppid ():**

```
int getpid (); int getppid ();
```

Разом з ідентифікатором процесу кожному процесу в ОС UNIX ставиться у відповідність також ідентифікатор групи процесів. До групи процесів об'єднуються всі процеси, які є процесами-нащадками одного і того ж процесу. Організація нової групи процесів виконується системним викликом **getpgrp (),** а отримання власного ідентифікатора групи процесів - системним викликом **getpgrp ().** Їх формат:

```
int setpgrp (); int getpgrp ();
```

З практичної точки зору в більшості випадків в рамках породженого процесу завантажується для виконання програма, визначена одним із системних викликів **execl (), execv (),** ... Кожен з цих системних викликів здійснює зміну програми, що визначає функціонування даного процесу:

```
execl (name, arg0, arg1, ..., argn, 0) char * name,
* arg0, * arg1, ..., * argn; execv (name, argv) char *
name, * argv []; execle (name, arg0, arg1, ..., argn,
0, envp) char * name, * arg0, * arg1, ..., * argn, *
envp []; execve (name, argv, envp) char * name, * arg
[], * envp [];
```

Сигнали - це програмний засіб, за допомогою якого може бути перервано функціонування процесу в ОС UNIX. Механізм сигналів дозволяє процесам реагувати на різні події, які можуть відбутися в ході функціонування процесу всередині нього самого або в зовнішньому світі. Кожному сигналу ставляться у відповідність номер сигналу і строкова константа, використовувана для осмисленої ідентифікації сигналу. Цей взаємозв'язок відображена у файлі описів **signal.h.** Для посилки сигналу використовується системний виклик, який має формат:

```
void kill (int pid, int sig);
```

У результаті здійснення такого системного виклику сигнал, специфікований аргументом **sig,** буде посланий процесу, який має ідентифікатор **pid.** Якщо **pid** не перевищує 1, сигнал буде посланий цілій групі процесів.

Використання системного виклику **signal ()** дозволяє процесу самостійно визначити свою реакцію на отримання тієї чи іншої події (сигналу):

```
int sig; int (* func) (); int * signal (sig, func) ();
```

Реакцією процесу, що здійснив системний виклик **signal ()** з аргументом **func,** при отриманні сигналу **sig** буде виклик функції **func ().**

Системний виклик **pause ()** дозволяє призупинити процес до тих пір, поки не буде отриманий якої-небудь сигнал:

```
void pause ();
```

Системний виклик **alarm (n)** забезпечує посилку процесу сигналу **SIGALARM** через **n** секунд.

В ОС UNIX існує спеціальний вид взаємодії між процесами - програмний канал. Програмний канал створюється за допомогою системного виклику **pipe (),** формат якого:

```
int fd [2]; pipe (fd);
```

Системний виклик **pipe ()** повертає два дескриптора файлу: один для запису даних в канал, інший - для читання. Після цього всі операції передачі даних виконуються за допомогою системних викликів введення-виведення **read / write.** При цьому система введення-виведення забезпечує призупинення процесів, якщо канал заповнений (при записі) або порожній (при читанні). Таких програмних каналів процес може встановити декілька. Відзначимо, що встановлення зв'язку через програмний канал спирається на спадкування файлів. Взаємодіючі процеси повинні бути спорідненими.

Завдання до лабораторної роботи

1. Розробити програму, що реалізовує дії, зазначені в завданні до лабораторної роботи з урахуванням наступних вимог:

- всі дії, що відносяться як до батьківського процесу, так і до породженим процесам, виконуються в рамках одного виконуваного файлу;
- обмін даними між процесом-батьком і процесом-нащадком пропонується виконати за допомогою тимчасового файлу: процес-батько після породження процесу-нащадка постійно опитує тимчасовий файл, чекаючи появи в ньому інформації від процесу-нащадка;
- якщо процесів-нащадків кілька, і всі вони готують деяку інформацію для процесу-батька, кожен з процесів поміщає у файл деяку структуровану запис, при цьому в цій структурованої записи містяться відомості про те, який процес посилає запис, і сама підготовлена інформація.

2. Модифікувати раніше розроблену програму з урахуванням таких вимог:
   - дії процесу-нащадка реалізуються окремою програмою, що запускається по одному із системних викликів **`execl (), execv ()`** і т. д. з процесу - нащадка;
   - процес-нащадок, після породження, повинен починати і завершувати своє функціонування за сигналом, що посилається процесом-предком (це ж відноситься і до декільком процесам-нащадкам);
   - обмін даними між процесами необхідно здійснити через програмний канал.

Варіанти завдань

1. Розробити програму, яка обчислює інтеграл на відрізку **`[A; B]`** від функції **`exp(x)`** методом трапецій, розбиваючи інтервал на **`K`** рівних відрізків. Для знаходження **`exp(x)`** програма повинна породити процес, що обчислює її значення шляхом розкладання в ряд по формулами обчислювальної математики.

2. Розробити програму, яка обчислює значення **`f (x)`** як суму ряду від **`k=0`** до **`k=N`** від вираження **`x^(2k+1)/(2k+1)!`** для значень **`x`**, рівномірно розподілених на інтервалі **`[0; Pi]`**, іщо виводить отриманий результат **`f (x)`** в файл в довічним форматі. У це час попередньо підготовлений процес - нащадок читає дані з файлу, перетворює їх в текстову форму і виводить на екрандо тих пір, поки процес-предок НЕ передасть йому через файл ключове слово (наприклад, "STOP"), що свідчить про закінченні процесів.

3. Розробити програму, яка обчислює щільність розподілу Пуассона з параметром **`lambda`** в точці **`k`** (**k** - ціле) по формулою **`f (k) = lambda ^ k * exp (-lambda) / k!`**. Для знаходження факторіала і **`exp (-lambda)`** програма повинна породити два паралельних процесу, що обчислюють ці величини шляхом розкладання в ряд по формулами обчислювальної математики.

4. Розробити програму, яка обчислює щільність опуклого розподілу в точці **`x`** по формулою **`f (x) = (1-cos (x)) / (Pi * x ^ 2)`**. Для знаходження **`Pi`** і **`cos (x)`** програма повинна породити два паралельних процесу, щобчислюють ці величини шляхом розкладання в ряд по формулами обчислювальної математики.

5. Розробити програму, яка обчислює значення щільності лог-нормального розподілу в точці **x (x> 0)** по формулою **f (x) = (1/2) * exp (- (1/2) * ln (x) ^ 2) / (x * sqrt (2 * Pi)).** Для знаходження **Pi, exp (x)** і **ln (x)** програма повинна породити три паралельних процесу, що обчислюють ці величини шляхом розкладання в ряд по формулами обчислювальної математики.

6. Розробити програму, яка обчислює число розміщень **n** елементів по **r** осередкам **N = n! / N (1)! * N (2)! * ... * N (r)!,** Яке задовольняє вимогу, що в осередок з номером **i** потрапляє рівно **n (i)** елементів **(i = 1..r)** і **n (1) + n (2) + ... + n (r) = n.** Для обчислення кожного факторіала необхідно породити процес - нащадок.

7. Розробити програму, яка обчислює число сполучень **C(k,n)=n!/(k!*(n-k)!).** Для обчислення факторіалів **n!, k!, (nk)!** повинні бути породжені три паралельних процесу - нащадка.

8. Розробити програму, яка обчислює значення **f (x)** як суму ряду від **k=1** до **k=N** від вираження **(-1)^(k+1)*x^(2k-1)/(2k-1)!** для значень **x,** рівномірно розподілених наінтервалі **[0; Pi],** і що виводить отриманий результат **f(x)** в файл в довічним форматі. У це час попередньо підготовлений процес-нащадок читає дані з файлу, перетворює їх в текстову форму івиводить на екран до тих пір, поки процес - предок НЕ передасть йому через файл ключове слово (наприклад, "STOP"), що свідчить про закінчення процесів.

9. Розробити програму, яка обчислює щільність нормального розподілу в точці **x** по формулою **(x)=exp(-x^2/2)/sqrt(2*Pi).** Для знаходження **Pi** і **exp (-x^2/2)** програма повинна породити два паралельних процесу, щообчислюють ці величини шляхом розкладання в ряд по формулами обчислювальної математики.

10. Розробити програму, яка обчислює інтеграл в діапазоні від 0 до 1 від підінтегральної вираження **4*dx/(1+x^2)** з допомогою послідовності рівномірно розподілених на відрізку **[0;1]** випадкових чисел, яка генерується процесом нащадком паралельно. Процес нащадок повинен завершитися після заздалегідь заданого числа генерацій **N.**

Контрольні питання

1. Яким чином може бути породжений новий процес? Яка структура нового процесу?

2. Якщо процес-предок відкриває файл, а потім породжує процес-нащадок, а той, у свою чергу, змінює положення покажчика читання-запису файлу, то чи зміниться положення покажчика читання-запису файлу процесу-батька?

3. Що станеться, якщо процес-нащадок завершиться раніше, ніж процес-предок здійснить системний виклик **wait ()?**

4. Чи можуть споріднені процеси розділяти загальну пам'ять?

5. Який алгоритм системного виклику **fork ()?**

6. Яка структура таблиць відкритих файлів, файлів і описувачів файлів після створення процесу?

7. Який алгоритм системного виклику **`exit ()`?**

8. Який алгоритм системного виклику **`wait ()`?**

9. У чому різниця між різними формами системних викликів типу **`exec ()`?**

10. Для чого використовуються сигнали в ОС UNIX?

11. Які види сигналів існують в ОС UNIX?

12. Для чого використовуються канали?

13. Які вимоги пред'являються до процесів, щоб вони могли здійснювати обмін даними за допомогою каналів?

14. Який максимальний розмір програмного каналу і чому?

Лабораторна робота № 2.6

КЕРУВАННЯ ПРОЦЕСАМИ

Мета лабораторної роботи – дослідити керування процесами ОС UNIX через системні виклики інтерфейсу прикладного програмування API та дослідити використання міжпроцесних сигналів UNIX за допомогою системних викликів.

Завдання

1. Дослідити системні виклики, що використовуються для керування процесами ОС.
2. Дослідити системні виклики, які використовуються для міжпроцесних сигналів.
3. Написати мовою С або Java та протестувати програми.

Варіанти

1. Процес породжує процес у разі введення значення користувачем. Новий процес має приймати це значення, виводити його на консоль та завершатися.

2. Процес породжує новий процес кожні 10 секунд. Новий процес повинен видавати повідомлення про своє породження та завершатися.

3. Процес породжує процес і передає йому термін „життя”. Новий процес має завершатися після цього терміну з видачею повідомлення.

4. Процес породжує 5 процесів; парні процеси повинні „жити” 5 секунд, непарні – 10, з видачею повідомлень про їх створення та завершення.

5. Процес породжує ланцюжок у дереві процесів з трьох процесів. Перевірити роботу програми за допомогою команди *ps*.

6. Процес породжує новий процес. Новий процес має приймати сигнал від процесу - предка, виводити на екран повідомлення і завершатися.

7. Процес породжує процес кожні 10 секунд. Новий процес повинен видавати повідомлення про своє породження та завершатися в разі прийомі сигналу.

8. Процес породжує процес. Якщо новий процес отримує сигнал, він має породжувати ще один процес.

9. Процес породжує 6 процесів, після чого парні процеси знищуються сигналом від предка.

10. Процес породжує ланцюжок з п’яти процесів. При подачі сигналу від предка, кожний з породжених процесів повинен надсилати сигнал нашадкам і завершатися.

Теоретичні відомості

Породження процесів в UNIX здійснюється за допомогою системного виклику *fork()*, який створює точну копію процесу, що породив даний. При цьому, *fork* повертає у батьківський процес значення ідентифікатору *PID* нового процесу, а в новий процес – 0. Завантажити в адресний простір процесу новий код можна, використовуючи системний виклик *exec()*. Завершити процес можна за допомогою системного виклику *exit()*. Прийом параметрів у процес здійснюється через аргументи *argc* та *argv*, які вміщують кількість і значення параметрів відповідно.

Текст програми *C* у ОС можна створити будь-яким текстовим редактором і зберегти в текстовий файл з розширенням *.c* (наприклад *prog.c*). Виклик

компілятора здійснюється з командного рядка командою *cc*. (Наприклад *cc prog.c*). Трансльований код зберігається у файл *a.out*. Файл *a.out* можна перейменувати та зробити таким, що виконується.

Сигнали є механізмом асинхронної взаємодії процесів. Кожен процес може приймати від іншого нумеровані сигнали, виконуючі у відповідь певну дію. Посилка сигналів здійснюється системним викликом *kill()*, який за замовчуванням приводить до завершення процесу – приймачу сигналу. У командній оболонці існує відповідна команда kill, за допомогою якої можна відправити процесу сигнал від процесу shell, наприклад: *kill 34586 –s 9* (надіслати сигнал з номером 9 процесу з ідентифікатором 34586)

Процес певного користувача реагує на сигнал тільки якщо він надійшов від процесу адміністратора або процесу того ж користувача.

Реакція певного процесу на сигнал з заданим номером може бути перевизначена шляхом виконання цим процесом системного виклику *signal()*. Параметрами цього виклику є номер сигналу, реакція на який перевизначається, та посилання (адрес) нової процедури обробки сигналу.

Контрольні питання:

1. Яким системним викликом створюється новий процес в Unix?
2. Як запустити в Unix програму з файла, що виконується?
3. Як можна передати аргументи до програми при запуску?
4. Які системні виклики дозволяють завершити або призупинити процес?
5. Як можна розрізнити батьківський та породжений процес при виконанні програми?
6. Що таке механізм сигналів?
7. Яким системним викликом здійснюється відсилання сигналу?
8. Які існують типи реакції на сигнал?
9. Як можна перевизначити реакцію процесу на сигнал?
10. Які користувачі мають право відсилати сигнали до процесів?

Лабораторна робота № 2.7

СИНХРОНІЗАЦІЯ ПРОЦЕСІВ

Мета роботи

Практичне освоєння механізму синхронізації процесів та їх взаємодії за допомогою програмних каналів.

Зміст роботи

1. Ознайомитися із завданням до лабораторної роботи.
2. Вибрати набір системних викликів, які забезпечують вирішення завдання.
3. Скласти програми на мові Сі, що реалізує необхідні дії.
4. Налагодити і від тестувати складені програми, використовуючи інструментарій ОС UNIX.
5. Захистити лабораторну роботу, відповівши на контрольні питання.

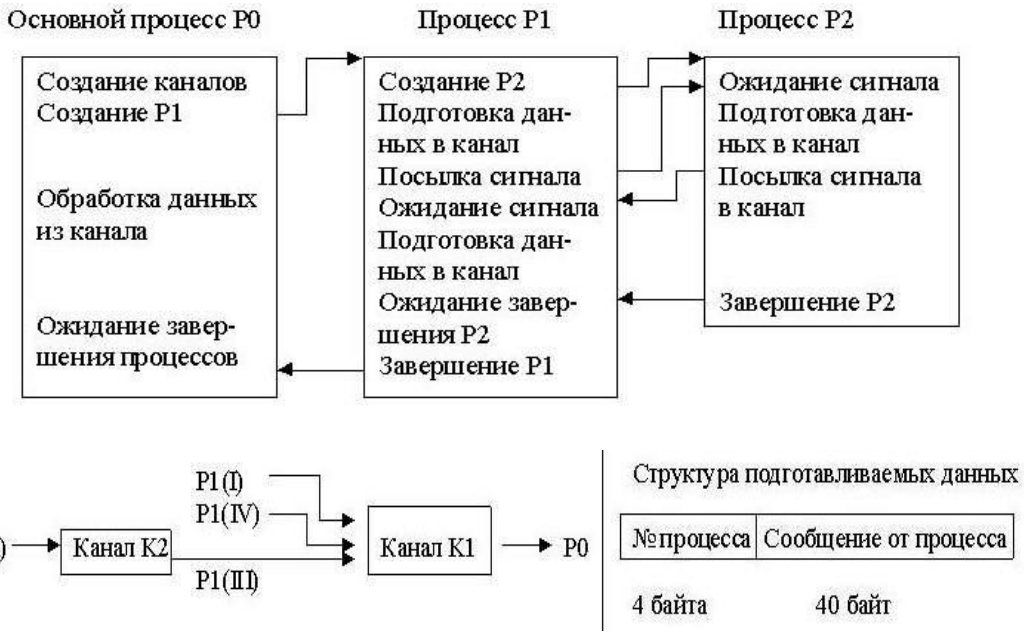
Методичні вказівки до лабораторної роботи

У попередній лабораторній роботі були розглянуті різні програмні засоби, пов'язані зі створенням і управлінням процесами в рамках ОС UNIX. Дана лабораторна робота передбачає комплексне їх використання при вирішенні задачі синхронізації процесів та їх взаємодії за допомогою програмних каналів. Коротко перерахуємо склад системних викликів, необхідних для виконання даної лабораторної роботи:

1. Створення, завершення процесу, отримання інформації про процес, - **`fork ()`, `exit ()`, `getpid ()`, `getppid ()`.**
2. Синхронізація процесів - **`signal ()`, `kill ()`, `sleep ()`, `alarm ()`, `wait ()`, `pause ()`.**
3. Створення інформаційного каналу і робота з ним - **`pipe ()`, `read ()`, `write ()`.**

Варіанти завдань

1. Вихідний процес створює два програмних каналу К1 і К2 і породжує новий процес Р1, а той, у свою чергу, ще один процес Р2, кожен з яких готує дані для обробки їх основним процесом. Підготовлювані дані процес Р1 поміщає в канал К1, а процес Р2 в канал К2, звідки вони процесом Р1 копіюються в канал К1 і доповнюються новою порцією даних. Схема взаємодії процесів, порядок передачі даних в канал і структура підготовлюваних даних показані нижче:



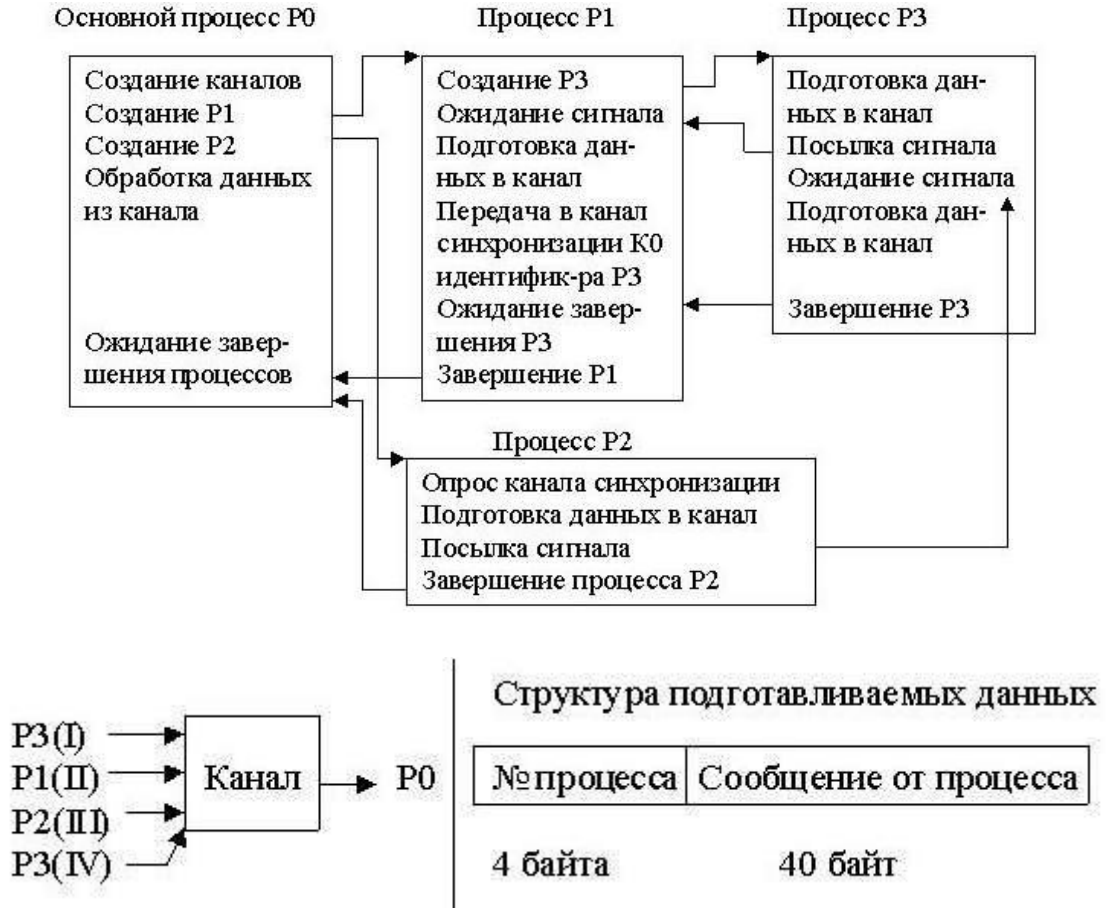
Обробка даних основним процесом полягає в читанні інформації з програмного каналу К1 і друку її. Крім того, за допомогою видачі повідомлень необхідно інформувати про всі етапи роботи програми (створення процесу, завершення посилки даних в канал і т.д.).

2. Вихідний процес створює програмний канал К1 і породжує два процеси Р1 і Р2, кожен з яких готує дані для обробки їх основним процесом. Підготовлені дані послідовно поміщаються процесами-синами в програмний канал і передаються основного процесу. Схема взаємодії процесів, порядок передачі даних в канал і структура підготовлюваних даних показані нижче:

Обробка даних основним процесом полягає в читанні інформації з програмного каналу і друку її. Крім того, за допомогою видачі повідомлень необхідно інформувати про всі етапи роботи програми (створення процесу, завершення посилки даних в канал і т.д.).

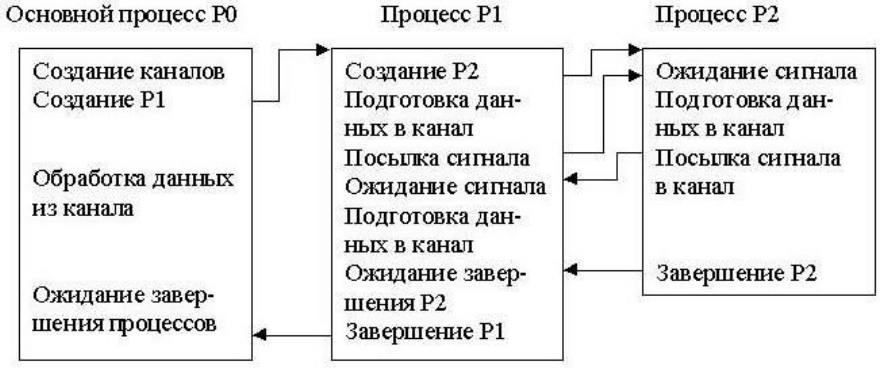
3. Вихідний процес створює програмний інформаційний канал К1, канал синхронізації К0 і породжує два процеси Р1 і Р2, з яких один (Р1) породжує ще один процес Р3. Призначення всіх трьох породжених процесів - підготовка даних для обробки їх основним процесом. Підготовлені дані послідовно поміщаються процесами-синами в програмний канал К1 і передаються основного процесу. Крім того, процес Р1 через канал синхронізації К0 повідомляє процесу Р2 ідентифікатор

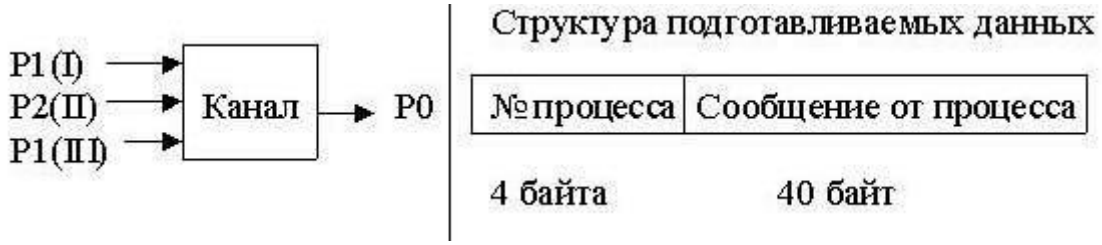
процесу Р3 з тим, щоб процес Р2 міг послати процесу Р3 сигнал. Схема взаємодії процесів, порядок передачі даних в канал і структура підготовлюваних даних показані нижче:



Обробка даних основним процесом полягає в читанні інформації з програмного каналу і друку її. Крім того, за допомогою видачі повідомлень необхідно інформувати про всі етапи роботи програми (створення процесу, завершення посилки даних в канал і т.д.).

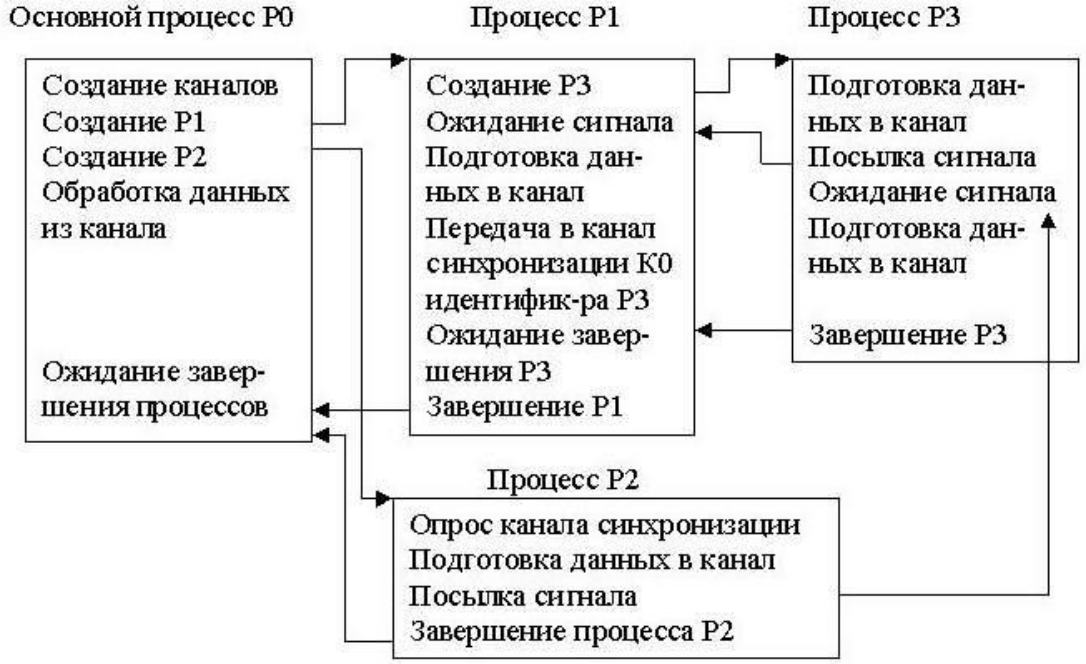
4. Вихідний процес створює програмний канал К1 і породжує новий процес Р1, а той, у свою чергу, ще один процес Р2, кожен з яких готує дані для обробки їх основним процесом. Підготовлені дані послідовно поміщаються процесами-синами в програмний канал і передаються основного процесу. Схема взаємодії процесів, порядок передачі даних в канал і структура підготовлюваних даних показані нижче:

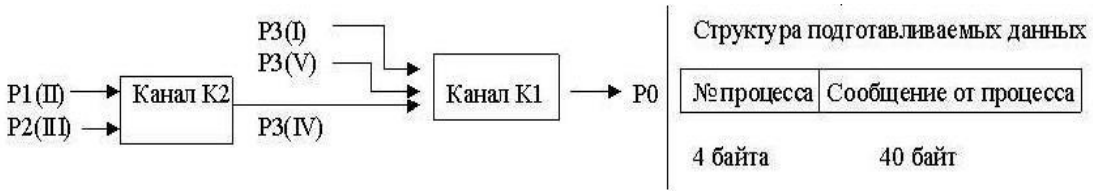




Обробка даних основним процесом полягає в читанні інформації з програмного каналу і друку її. Крім того, за допомогою видачі повідомлень необхідно інформувати про всі етапи роботи програми (створення процесу, завершення посилки даних в канал і т.д.).

5. Вихідний процес створює два програмних інформаційних каналу К1 і К2, канал синхронізації К0 і породжує два процеси Р1 і Р2, з яких один (Р1) породжує ще один процес Р3. Призначення всіх трьох породжених процесів - підготовка даних для обробки їх основним процесом. Підготовлювані дані процес Р3 поміщає в канал К1, а процеси Р1 і Р2 в канал К2, звідки вони процесом Р3 копіюються в канал К1 і доповнюються новою порцією даних. Крім того, процес Р1 через канал синхронізації К0 повідомляє процесу Р2 ідентифікатор процесу Р3 з тим, щоб процес Р2 міг послати процесу Р3 сигнал. Схема взаємодії процесів, порядок передачі даних в канал і структура підготовлюваних даних показані нижче:

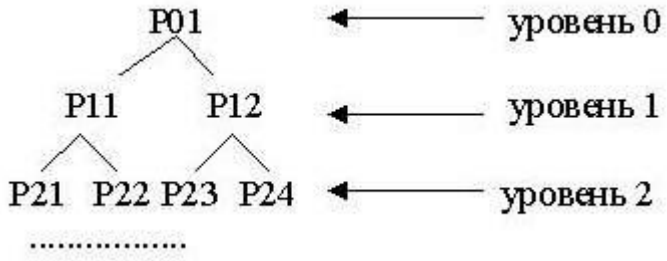




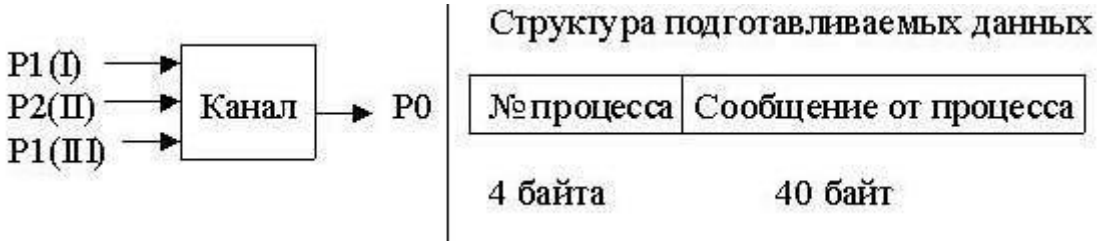
Обробка даних основним процесом полягає в читанні інформації з програмного каналу К1 і друку її. Крім того, за допомогою видачі повідомлень необхідно

інформувати про всі етапи роботи програми (створення процесу, завершення посилки даних в канал і т.д.).

6. Програма породжує ієрархічне дерево процесів. Кожен процес виводить повідомлення про початок виконання, створює пару процесів, повідомляє про це, чекає завершення породжених процесів і потім закінчує роботу. Оскільки дії в рамках кожного процесу однотипні, ці дії повинні бути оформлені окремою програмою, що завантажується системним викликом **exec ()** . Параметр програми - число рівнів (НЕ більше 5).



7. Вихідний процес створює програмний канал К1 і породжує новий процес Р1, а той, у свою чергу, породжує ще один процес Р2. Підготовлені дані послідовно поміщаються процесами-синами в програмний канал і передаються основного процесу. Файл, що читається процесом Р2, має бути досить великий з тим, щоб його читання не завершилося раніше, ніж закінчиться встановлена затримка в n секунд. Після спрацьовування будильника процес Р1 посилає сигнал процесу Р2, перериваючи читання файлу. Схема взаємодії процесів, порядок передачі даних в канал і структура підготовлюваних даних показані нижче:



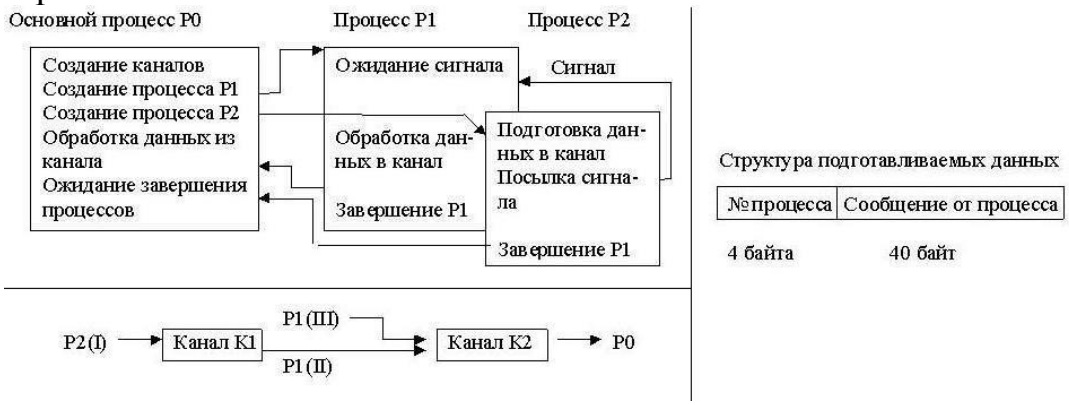
Обробка даних основним процесом полягає в читанні інформації з програмного каналу і друку її. Крім того, за допомогою видачі повідомлень необхідно

інформувати про всі етапи роботи програми (створення процесу, завершення посилки даних в канал і т.д.).

8. Вихідний процес створює програмний канал К1 і породжує два процеси Р1 і Р2, кожен з яких готує дані для обробки їх основним процесом. Підготовлені дані послідовно поміщаються процесами-синами в програмний канал і передаються основного процесу. Схема взаємодії процесів, порядок передачі даних в канал і структура підготовлюваних даних показані нижче:

Обробка даних основним процесом полягає в читанні інформації з програмного каналу і друку її. Крім того, за допомогою видачі повідомлень необхідно інформувати про всі етапи роботи програми (створення процесу, завершення посилки даних в канал і т.д.).

9. Вихідний процес створює два програмних каналу К1 і К2 і породжує два процеси Р1 і Р2, кожен з яких готує дані для обробки їх основним процесом. Підготовлювані дані процес Р2 поміщає в канал К1, потім вони звідти читаються процесом Р1, переписуються в канал К2, доповнюються своїми даними. Схема взаємодії процесів, порядок передачі даних в канал і структура підготовлюваних даних зображені нижче:



Обробка даних основним процесом полягає в читанні інформації з програмного каналу К2 і друку її. Крім того, за допомогою видачі повідомлень необхідно інформувати про всі етапи роботи програми (створення процесу, завершення посилки даних в канал і т.д.).