Лабораторна робота № 1.
Управління файловою системою та методи мультиплексованого вводу-виводу.

Завдання

***Частина 1: Функції послідовного вводу-виводу**. Скласти програму, яка:*

1. Отримує аргументами командної строки два імені файлів
2. Перший файл відкриває для читання, другий до перезапису, або створює, якщо його не існує (для створення використати атрибути 0644)
3. Послідовно читає дані з першого файлу буферами фіксованого об'єму (наприклад 512 байт) та записує у другий.
4. Перед записом із вмістом буферу проводить таке перетворення – усі рядкові літери латинського алфавіту перетворює на відповідні прописні. Для цього розрахувати відповідний зсув за таблицею ASCII (див. man ascii).
5. В процесі перезапису проводить підрахунок перезаписаних байт, та наприкінці виводить у вихідний потік сумарний обсяг переписаних даних.

***Частина 2: Функції мультиплексованого вводу-виводу**. Скласти програму, яка:*

1. Отримує аргументом командної строки довільний строковий ідентифікатор.
2. Налаштовує системний виклик *select* для очікування читання у вхідному потоці (файловий дескріптор STDIN_FILENO) з таймаутом 5 секунд.
3. При отриманні можливості читання, прочитати з потоку буфер довжиною не більше 1024 байта та вивести його у вихідний потік з поміткою у вигляді ідентифікатора п.1.
4. При спливанні таймауту, вивести у потік помилок повідомлення про це з поміткою у вигляді ідентифікатора п.1 та знову налаштувати системний виклик *select* (п.2).
5. Протестувати роботу програми отримуючи через вхідний потік результати вводу з клавіатури.

Рекомендації до виконання.

- Для перезапису файлів використовувати виключно системні виклики. Для виводу у вихідний потік припустимо використання функцій <*stdio*>
- Обов'язково діагностувати помилки, що виникають під час відкриття файлів, читання та запису.
- Для діагностики помилок використовувати *errno* та *strerror*
- Для налаштування системного виклику *select* використати макроси FD_ZERO, FD_SET та структуру timeval.
- **Додаткове завдання:** Повторити функціонал програми ч.2. з використанням системного виклику *poll.*

Посилання

- Рекомендована література: №№ 1-3.
- Офіційна документація стандартної бібліотеки Сі 2-й розділ, статті: open, creat, close, read, write, fsync, fseek, select, poll, errno, strerror.

Лабораторна робота № 2.
Управління процесами, відокремлення процесів.

Завдання

Частина 1. Запуск, завершення та параметри процесу. *Скласти програму, яка:*

1. Отримує та друкує інформацію про параметри свого процесу за допомогою системних викликів getpid(), getgid(), getsid() тощо.
2. Виконує розгалудження процесу за допомогою системного виклику fork().
3. Для процесу-батька та процесу-нащадка окремо роздруковує їхні ідентифікатори у циклі.
4. Виконати очікування процесом-батьком завершення нащадка.
5. Повідомляє про завершення процесів — батька та нащадка.
6. Пояснити отримані результати

Частина 2. Демонізація процесу. *Скласти програму, яка:*

1. Відкриває на запис текстовий файл логу та робить у нього запис про старт програми.
2. Виконує демонізацію свого функціоналу за таким алгоритмом:
    - Виконує fork()
    - Для процесу-батька робить запис у лог про породження нащадка та закриває себе викликом exit().
    - Для поцесу-нащадка:
        - робить setsid()
        - змінює поточний каталог на “/”
        - закриває усі відкриті батьком дескриптори
        - відкриває “/dev/null” на запис для трьох стандартних потоків
3. Відкриває лог та робить у нього запис про параметри демонізованого процесу.
4. Виконує витримку часу у нескінченному циклі.
5. Пояснити отримані результати

Рекомендації до виконання.

1. Розділення функціоналів батька та нащадка виконати за результатом виконання fork().
2. Для очікування завершення нащадка, використовувати системний виклик wait().
3. Для стеження за останніми записами у логу використовуйте tail -f
4. Для завершення роботи демона використовуйте kill -9

Посилання.

- Рекомендована література: №№ 1-3.
- Офіційна документація стандартної бібліотеки Сі 2-й розділ, статті:
  getpid, getgid, getuid, getgid, fork,.wait, exit, setsid.

Лабораторна робота № 3.
Сигнали. Розділення пам`яті між процесами.

Завдання.

***Частина 1. Обробка сигналів**. Скласти програму, яка:*

1. Описує глобальний дескриптор файла логу.
2. Описує функцію-обробник сигналів, прототипу

```
void signal_handler( int signo, siginfo_t *si, void * ucontext );
```

3. У функції-обробнику виконати запис у файлі логу з докладним розкриттям структури `siginfo_t`, яка подана на вхід.
4. Відкриває файл логу на запис.
5. Відмічає в ньому факт власного запуску та свій pid.
6. Описує структуру sigaction, у якій вказує на функцію обробник.
7. Реєструє обробник для сигналу SIGHUP із збереженням попереднього обробника.
8. Переходить до нескінченного циклу із засинанням на кілька секунд та відмітками у файлі логу.
9. Протестувати отриману програму, посилаючи до неї сигнали утілітою kill, та спостерігаючи результат у файлі логу.
10. Пояснити отримані результати

***Частина 2. Розподілена пам`ять**. Скласти програму, яка:*

1. Описує структуру датума, яка містить ціле значення для ідентифікатора процесу, ціле значення для мітки часу та строку фіксованої довжини.
2. Реєструє обїект розподіленої пам`яті через виклик shm_open
3. Приводить його до розміру кратного розміру структури датуму
4. Відображае отриманий об`єкт у пам`ять через показчик на структуру датуму та виклик mmap
5. Переходить до нескінченного циклу у якому:
    a. Запитує строку з клвіатури
    b. Вичитує та презентує вміст структури датуму
    c. Записує у структуру натомість свій ідентификатор процесу, поточний час та отриману строку
6. Протестувати отриману програму, запустивши два її примірники у різних сесіях.

Рекомендації до виконання.

1. Аргумент ucontext у обробнику сигналів може бути проігнорований
2. До параметру sa_flags структури sigaction додати флаг SA_SIGINFO
3. Для приведення розміру розподіленого сегменту використовуйте ftruncate
4. Перед читанням структури датуму з розподіленого сегменту, використовуйте msync
5. Для стеження за останніми записами у логу використовуйте tail -f

Посилання.

- Рекомендована література: №№ 1-3.
- Офіційна документація стандартної бібліотеки Сі 2-й розділ, статті: signal, kill, raise, sigaction, sigqueue, mmap, shm_open, shm_unlink.

Лабораторна робота № 4.
Мережеві сокети. Елементарний сервер масового обслуговування.

Завдання.

Частина 1. Побудова мережевого сервера. *Скласти програму, яка:*

1. Виконує демонізацію, подальший функціонал стосується демона.
2. Описує глобальний дескриптор файла логу.
3. Описує структуру *struct sockaddr_in* з параметрами: формат сокетів - PF_INET, адреса - будь яка, порт - 3200 + номер варіанта.
4. Формує сокет типу SOCK_STREAM формату PF_INET.
5. Налаштовує сокет на очікування запитів за допомогою *bind.*
6. Запускає нескінченний цикл обробки запитів від клієнтів.
7. На кожний запит виконує fork для породження процесу обробки.
8. Батьківський процес закриває сокет.
9. Процес обробки в нескінченному циклі отримує від клієнта строки за допомогою *recv*, додає до них свій префікс у вигляді поточного часу та власного *pid*, та повертає клієнту за допомогою *send*.
10. Процес обробки завершує обробку даних від клієнта отримавши строку "close".
1. Кожна дія сервера супроводжується відміткою у логу.

Частина 2. Побудова мережевого клієнта. *Скласти програму, яка:*

7. Описує структуру *struct sockaddr_in* з параметрами: формат сокетів - PF_INET, адреса - обчислена за викликом *htonl(INADDR_LOOPBACK)*, порт - 3200 + номер варіанта.
8. Формує сокет типу SOCK_STREAM формату PF_INET.
9. Налаштовує сокет на підключення до сервера за допомогою *connect.*
10. В нескінченному циклі запитує рядки від оператора, передає їх на сервер та друкує отримані відповіді.
11. Робота закінчується після вводу оператором рядка "close" та отримання на нього відповіді від сервера.

Частина 3. Дослідити роботу мережевого сервера.

Дослідження провести запустивши сервер та кілька примірників клієнта у різних сесіях. Проаналізувати лог сервера та зробити висновки щодо моменту запуску та закриття процесів обробки з'єднань.

Рекомендації до виконання.

1. Порти сокетів формувати за допомогою htons.
2. Адреси сокетів формувати за допомогою htonl.
3. Обробляти помилки при виклику socket, bind, accept, connect. Виводити діагностику.
4. Додатково вивчити принцип побудови серверів на неблокуючих сокетах (O_NONBLOCK) з використанням select.
5. Для стеження за останніми записами у логу використовуйте tail -f

Посилання.

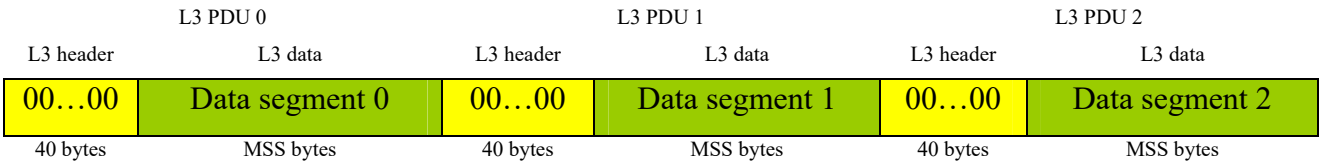
- Рекомендована література: №№ 1-3.
- Офіційна документація стандартної бібліотеки Сі 2-й розділ, статті: socket, fcntl, bind, listen, connect, accept, send, recv.

Лабораторна робота № 5.
Реалізація 4-рівневого протоколу передачі даних.

Завдання.

Скласти модульну програму, яка реалізує 4 рівні протоколу передачі даних. Кожен рівень має бути реалізований щонайменше однією окремою функцією (layer1 .. layer4). Кожен рівень має отримувати буфер даних від попереднього рівня та передавати їх наступному. Дані, сформовані першим рівнем протоколу, мають бути передані через сокет FIFO до іншого екземпляра програми та розгорнуті у зворотньому порядку.

1. Крім обробників рівнів протоколу, у програмі мають бути реалізовані функціонали transmit та receive. Кожен екземпляр програми має бути запущений у режимі прийому або передачі.
2. Програма має приймати наступні аргументи командного рядка:
   2.1. Режим роботи: -t для передачі або –r для прийому
   2.2. Щлях до сокету FIFO (за відсутності створити)
   2.3. Шлях до файлу даних (для передавача має існувати та мати розмір щонайменше 2Мб, для приймача має бути створений чи перезаписаний)
3. Функціонал ***transmit*** передбачає:
   3.1. Функція layer4 має завантажити дані з файлу та передати їх у вгляді буфера до наступного рівня протоколу (layer3). Розмір буфера не має перевищувати 2кб.
   3.2. Завданням функції третього рівня протоколу є формування буферу передачі, який містить дані, розбиті на блоки (portable data units, PDU) відповідного рівня, до яких кожен рівень протоколу додає специфічні для нього заголовки.
   3.3. Буфер передачі має вміщувати рівно 40 PDU. Дані, отримані від layer4 мають бути розбиті на блоки, розмір яких не перевищує значення maximum segment size (MSS = 60).
   3.4. Буфер під передачу має бути виділений один раз. Не дозволяється копіювати дані з буфера в буфер. Кожна функція відповідного рівня може переставляти дані у межах буфера та додавати свої блоки.
   3.5. Кожному блоку має передувати заголовок 3-го рівня розміром у 40 байтів. Він має бути заповнений нулями:



3.6. Далі буфер має бути переданий функції другого рівня протоколу.

3.7. Другий рівень протоколу додає до L3PDU заголовок та суфікс.

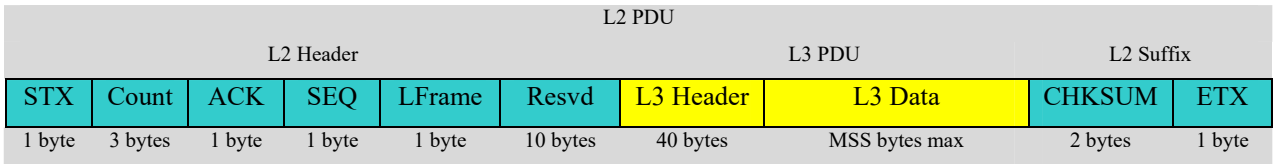
3.8. Заголовок повинен мати таку структуру:

- STX 1b =0x02
- COUNT 3b розмір повного пакету (тут 120 байт крім останнього)
- ACK 1b =0
- SEQ 1b номер пакету у послідовності
- LFRAME 1b =0 для усіх пакетів крім останнього, для нього =0x01 для останнього в буфері або =0x0F для останнього у вхідному файлі
- RESVD 1b =0

3.9. Суфікс повинен мати таку структуру:

- CHKSUM 2b контрольна сума пакету
- ETX 1b =0x03

3.10. Структура послідовності пакетів має набути такого вигляду:



3.11. Контрольна сума пакету має бути обчислена за алгоритмом CRC-16-IBM (з основою поліному 0x8005) на усіх байтах пакету, вважаючи на час обчислення байти контрольної суми нульовими.

3.12. Далі другий рівень протоколу має викликати функціонал першого рівня (layer1) для кожного пакету другого рівня окремо.

3.13. Функція першого рівня повинна отримати вказівник на поточний пакет другого рівня та його розмір, відкрити сокет FIFO на запис, отримати дозвіл на запис через select або poll, передати пакет приймачу, перейти в режим читання та отримати пакет підтвердження такої структури:

- STX 1b =0x02
- SEQ 1b номер послідовності переданного пакету
- ACK 1b =0x06 для коректного прийому або =0x15 для помилки прийому

- ETX 1b =0x03

3.14. У разі коректного прийому функція першого рівня має повернути 0 і функція другого рівня – викликати її дял наступного пекету у буфері.

3.15. У разі помилки прийому – функція першого рівня має повторити передачу пакету. У разі повторення помилки прийому більше TTR=16 разів має повернути -1 і функція другого рівня – викликати її повторно дял того самого пакету.

3.16. У разі повторення помилок на другому рівні більше TTR разів, програма передавач має завершитися із нотифікацією про критичну помилку передачі.

3.17. У разі неможливості відкриття сокету, помилки запису, помилки очікування poll або select чи відсутності пакету підтвердження, функція першого рівня також має повернути -1.

3.18. Поле LFrame у заголовку останнього пакета у буфері має дорівнювати 0x01

3.19. Після завершення передачі буфера програма має повернутися до п. 3.1 та вичитати з вхідного файлу наступний буфер.

3.20. Для останнього пакета останнього буфера поле LFrame має дорівнювати 0x0F.

3.21. Програма у режимі передачі має завершитися або через фатальну помилку, або після успішної передачі останнього пакету останнього буфера після вичерпання вхідного файлу.

4. Функціонал ***receive*** передбачає:

4.1. Виклик функції першого рівня у режимі прийому.

4.2. Функція першого рівня має відкрити сонет FIFO на читання та очікувати готовності даних.

4.3. По готовності, функція першого рівня має отримати пакет, перевірити його контрольну суму, за результатами перевірки відкрити сокет на запис та передати пакет підтвердження із відповідними значеннями SEQ та ACK.

4.4. Пакет має бути записаний до буфера прийому, під котрий має бути виділена пам'ять із розрахунку на 40 L2PDU.

4.5. Функція першого рівня має бути викликана для кожного наступного пакету окремо і отримати вказівник на відповідне зміщення у буфері прийому.

4.6. Ознакою завершення прийому є значення LFrame останнього отриманого пакету.

4.7. Після заповнення буфера прийому він має бути переданий до функції другого рівня.

4.8. Функція другого рівня має видалити з буфера заголовки та суфікси другого рівня та усунути пробіли зклеївши разом усі наявні L3PDU. Повернути результуючу довжину буфера.

4.9. Після ціого буфер має бути переданий до функції третього рівня.

4.10. Функція третього рівня має видалити заголовки третього рівня та усунути пробіли склеївши разом усі наявні блоки даних. Повернути результуючу довжину буфера.

4.11. Після цього буфер має отримати функція четвертого рівня. Вона повинна відкрити вихідний файл (для першого буфера з O_CREAT, O_TRUNCATE, для решти – з O_APPEND) та записати у нього буфер.

4.12. Далі, відповідно до значення LFrame останнього пакету у буфері, програма має або перейти до п. 4.1 для отримання нового буферу, або успішно завершитись.

5. Функції відповідних рівнів протоколу можуть бути реалізовані як окремо для прийому та передачі, так і у одній функції з аргументом, що вказує на режим.
6. При перевірці роботи програми буде врахована як безпомилкова передача даних між файлами, так і повнота обробки усіх можливих помилок.

Посилання.

- Рекомендована література: №№ 1-3.
- Офіційна документація стандартної бібліотеки C, статті: open,read,write,select,poll,mkfifo.

Лабораторна робота № 6.
Багатопотоковий HTTP сервер з пулом потоків

Завдання.

Скласти програму, що реалізує функціонал HTTP серверу з пулом потоків.

1. Програма має приймати наступні аргументи командного рядка:
   1.1. –p <PORT> - ціле число – номер порта
   1.2. –d <ROOT> - шлях до кореневого каталогу сервера
   1.3. –t <THREADS> - ціле число – кількість потоків обробки (< 16)
   1.4. –q <QUEUE> - ціле число – глибина черги з'єднань (< 128)
   1.5. –l <LOG> - Log файл

   Для розбору командного рядка використати getopt.
2. Програма має містити наступні глобальні об'єкти:
   2.1. Структуру connection_queue, яка реалізує функціонал черги з'єднань клієнтів із сокетом. Структура має містити зв'язаний список структур з'єднання, вказівники на його початок та кінець та м'ютекс для забезпечення потокової безпеки даних та семафор для організації сигналу «непуста черга». Кожна структура з'єднання має містити дані, що отримані від клієнта після виконання функції сокету accept та читання із сокету:
      2.1.1. файловий дескриптор клієнтського сокету
      2.1.2. вказівник на структуру sockaddr_in що містить клієнську адресу
      2.1.3. вказівник на структуру http_request що містить розібраний відповідно до https://tools.ietf.org/html/rfc7231#section-4.3.1 запит від клієнта (має бути підтриманий тільки метод GET) і має такий зміст:
         2.1.3.1. char method[8] = «GET»
         2.1.3.2. char *host
         2.1.3.3. char *path
         2.1.3.4. char *file
         2.1.3.5. struct timeval request_time
         2.1.3.6. struct timezone request_timezone

         Із запиту має бути розібраний перший рядок, перевірений метод, виділений шлях запиту, розбитий на шлях у файловій системі та ім'я файлу, відділені аргументи та фрагмент запиту (та проігноровані), знайдений рядок, що містить хост запиту,

виділений хост, решта запиту проігнорована. Якщо метод запиту не підтримуваний – має бути сформована помилка 503.

Для розбору запиту може бути використана бібліотека https://www.w3.org/Library/src/HTParse.html

Час та часовий пояс моменту запиту мають бути отримані через gettimeofday.

2.2. Структуру thread_pool, що реалізує множину потоків обробки з'єднань і містить: подвійний вказівник на pthread_t, що є масивом потоків, лічильник існуючих та зайнятих потоків та мьютекс для захисту лічильників. Також він має містити посилання на функцію обробника черги з'єднань та посилання на структуру черги.

3. Для обслуговування глобальних об'єктів мають бути творені такі функції:
   3.1. thread_pool_init - створення пулу, приймає кількість потоків
   3.2. thread_pool_destroy - завершує усі потоки та звільняє ресурси
   3.3. connection_queue_init - створює чергу
   3.4. connection_queue_push - вставляє структуру з'єднання у чергу
   3.5. connection_queue_pull - виймає структуру з'єднання з черги
   3.6. connection_queue_destroy - звільняє ресурси черги
4. Функціонування сервера має бути організовано таким чином:
   4.1. Основний потік демонізується.
   4.2. Створюється пул та черга запитів.
   4.3. Демон виконує bind, listen, та запускає цикл accept з беклогом у 5 з'єднння.
   4.4. Для кожного дескриптора, що отриманий через accept, виконується читання з сокету, розбор запиту, формування структури запиту та вставка структури до черги запитів через connection_queue_pull.
   4.5. Для коректного завершення програми необхідно зареєструвати обробники сигналів SIGPIPE та SIGTERM.
   4.6. При створенні пулу використовується функція обробки з'єднання, що приймає аргументом вказівник на чергу зє'днань.
   4.7. Функція обробки з'єднань у нескінченному циклі:
      4.7.1. Очікує готовності черги з'єднань за допомогою pthread_cond_wait.
      4.7.2. Отримує структуру з'єднання через connection_queue_pull
      4.7.3. Аналізує шлях (відносно кореневого каталогу сервера) та ім'я файлу запиту
      4.7.4. Якщо файл не вказаний, підставляє за замовчуванням «index.html»
      4.7.5. Перевіряє наявність та регулярність файлу
      4.7.6. Якщо шлях або файл не існує – формує помилку 403

4.7.7. Обчислює Content-type через суфікс імені файлу та таблицю з https://www.iana.org/assignments/media-types/media-types.txt (обрати декілька найбільш поширених типів, щонайменше htm,html,txt,jpg,jpeg,png,…)

4.7.8. Відкриває файл на читання та вичитує його у буфер.

4.7.9. Формує відповідь до клієнта (код 200) відповідно до https://www.w3.org/Protocols/rfc2616/rfc2616.html що містить наступні поля обов1'язково:

4.7.9.1. HTTP/1.1 200 Ok

4.7.9.2. Date:

4.7.9.3. Content-Type:

4.7.9.4. Content-Length:

4.7.9.5. буфер, вичитаний з файла

4.7.10. Посилає відповідь клієнту через дескриптор з'єднання та закриває його.

5. Для тестування серверу використовувати довільний веб-браузер.
6. Перевірити коректне відпрацювання помилки переповнення беклогу сокета та черги з'єднань. При виникненні таких помилок передати клієнту 503.
7. Передбачити debug режим, за яким писати в лог не тільки інформацію про помилкові ситуації, а й повний зміст отриманих запитів та переданий відповідей.

Додаткове завдання.

До структури з'єднання додати попередньо обчислений через stat розмір файлу.
Реалізувати у додатковому потоці планувальник, який би приоретизував чергу до виконання за алгоритмом Shortest Job First (SJF).

Посилання.

- Рекомендована література: №№ 1-3.
- Офіційна документація стандартної бібліотеки Сі 2-й розділ, статті:
  socket, fcntl, bind, listen, connect, accept, send, recv, pthread_create, pthead_join, pthread_mutex_lock, pthread_mutex_unlock, pthread_cond_wait, pthread_cond_signal, sem_wait, sem_post.

Перелік рекомендованої літератури

1. У.Стивенс "UNIX: Профессиональное программирование" "- СПб:Символ,2007
2. У.Стивенс "UNIX: Взаимодействие процессов" "- СПб:БХВ-Питер,2002
3. http://advancedlinuxprogramming.com/
4. Керниган Б.В., Пайк Р. UNIX - универсальная среда программирования: пер. с англ.- М.: Финансы и статистика,1992.-304 с.
5. Кейт Хэвиленд, Дайна Грэй, Бен Салама " Системное программирование в UNIX. Руководство программиста по разработке ПО": Пер. с англ.- М, ДМК Пресс,2000.
6. Теренс Чан " Системное программирование на C++ для UNIX.":Пер. с англ.-K:BHV, 1999.
7. Андрей Робачевский "Операционная система UNIX".- СПб:БХВ-Санкт-Петербург, 2000.
8. Бах Морис Дж. Архитектура операционной системы UNIX. //THE DESIGN OF THE UNIX OPERATING SYSTEM by Maurice J. Bach// Пер. с англ. к.т.н. Крюкова А.В. Copyright 1986 Корпорация Bell Telephone Laboratoties. Издано корпорацией Prentice-Hall. Отделение Simon & Schuster Энглвуд Клиффс, Нью-Джерси 07632. Серия книг по программному обеспечению издательства Prentice-Hall. Консультант Брайан В. Керниган.