시스템프로그래밍 과제 보고서

Proxy 3-2

수업 명: 시스템프로그래밍 월5수6

담당 교수: 김태석 교수님

학과: 컴퓨터정보공학부

학번: 2023202070

이름: 최현진

제출일: 2025.06.01

Introduction

이번 Proxy 3-2 과제는 Proxy 3-1 과제의 연장선으로, semaphore 기반의 동기화에 더해 POSIX Thread를 활용한 비동기 로그 기록 기능이 추가된 것이 핵심이다.

기존에는 자식 프로세스가 직접 logfile.txt에 접근하여 로그를 작성했지만, 이번 과제에서는 자식 프로세스가 생성한 thread가 로그 기록을 수행하도록 구현하여, 다중 자식 프로세스 환경에서 로그 접근 구조를 보다 유연하게 확장하였다.

특히 로그 기록은 이전과 동일하게 System V Semaphore를 통해 보호되며, 임계 구역(critical section) 진입 전과 종료 후에는 다음과 같은 터미널 메시지가 출력된다:

\*PID# XXXX is waiting for the semaphore.

\*PID# XXXX is in the critical zone.

\*PID# XXXX create the \*TID# XXXXX.

\*TID# XXXXX is exited.

\*PID# XXXX exited the critical zone.

HIT 및 MISS 판단 이후 로그 작성 시, 자식 프로세스는 동적으로 thread를 생성하여 로그 메시지를 전달하며, 생성된 thread는 로그 기록 후 종료를 출력한다. pthread\_join()을 통해 스레드 종료를 기다림으로써 로그 순서의 일관성을 보장하였다. 또한, 과제의 통일성과 구조적 일관성을 위해 부모 프로세스의 종료 로그 역시 thread를 생성하여 기록하도록 구현하였다.

이를 통해 이번 과제는 멀티프로세스-멀티스레드 환경에서의 공유 자원 접근 제어, thread 기반의 작업 위임, 임계 구역의 thread-safe한 처리 방식에 대한 이해를 실습적으로 심화시키는 데 목적이 있다.

Flow Chart

Pseudo code

|  |
| --- |
| main: |
| 프로그램 시작  시작 시간 저장  SIGINT 핸들러 등록  세마포어 초기화  부모 PID 저장  홈 디렉토리 경로 얻기  ~/logfile 디렉토리 생성  서버 소켓 생성  서버 주소 설정  bind()로 IP/PORT 할당  listen()으로 클라이언트 연결 대기  무한 반복:  accept()로 클라이언트 연결 수락  fork()로 자식 생성  자식이면:  서버 소켓 닫기  자식 시작 시간 저장  클라이언트 요청 수신  현재 시간 저장  method가 GET이 아니거나 URL이 없거나  .ico/.css/.txt/.firefox 포함되면 isURL = 0  SHA-1 해시 수행  앞 3자리 → 캐시 디렉토리 이름  뒤 37자리 → 캐시 파일 이름  ~/cache 디렉토리 생성  ~/cache/xxx 디렉토리 생성  디렉토리에서 캐시 파일 탐색  캐시 HIT이면:  isURL이면:  세마포어 wait  로그 메시지 생성  pthread\_create()로 log\_thread 실행  TID 출력  pthread\_join()으로 스레드 종료 대기  세마포어 post  캐시 파일 열기  캐시 데이터 브라우저로 전송  캐시 MISS이면:  isURL이면:  세마포어 wait  로그 메시지 생성  pthread\_create()로 log\_thread 실행  TID 출력  pthread\_join()으로 스레드 종료 대기  세마포어 post  요청에서 hostname, port, path 추출  origin 서버 IP 얻기  소켓 생성 및 origin 서버 연결  요청 라인 구성 (HTTP/1.0)  요청 헤더 정리 및 추가  origin 서버로 요청 전송  origin 서버로부터 응답 수신  응답을 브라우저로 전송  캐시 파일 생성 및 응답 저장  클라이언트 소켓 닫기  자식 종료  부모면:  자식 수 증가  클라이언트 소켓 닫기  waitpid로 좀비 프로세스 회수 |

|  |
| --- |
| init\_semaphore |
| semkey = PORT 번호  semget()로 세마포어 생성 (1개짜리 세트)  SETVAL 통해 초기값 1로 설정 |

|  |
| --- |
| wait\_semaphore |
| sembuf 구조체로 P 연산 (-1)  semop() 호출 → 세마포어 진입 (임계영역 진입) |

|  |
| --- |
| post\_semaphore |
| sembuf 구조체로 V 연산 (+1)  semop() 호출 → 세마포어 해제 (임계영역 탈출) |

|  |
| --- |
| log\_thread(void \*arg) |
| 홈 디렉토리 경로 얻기  ~/logfile 경로로 이동  logfile.txt 열기 (a 모드)  log 메시지 내용 기록  파일 닫기  TID 종료 메시지 출력  인자 메모리 해제  pthread\_exit() 호출 |

|  |
| --- |
| sig\_int(signo) |
| 현재 프로세스가 자식이면 바로 종료  종료 시각 저장  실행 시간 = 종료 시각 - 시작 시각  세마포어 wait  \*PID# 메시지 출력 (진입)  종료 로그 메시지 생성  pthread\_create()로 log\_thread 실행  \*TID# 생성 메시지 출력  pthread\_join()으로 스레드 종료 대기  \*PID# 메시지 출력 (탈출)  세마포어 post  세마포어 삭제  부모 프로세스 종료 |

결과 화면

1. proxy 3-2

|  |
| --- |
| 1. make |
| $ ls: Makefile과 소스 코드를 작성하였다.  $ make: make를 실행하여 컴파일을 수행하였고, gcc 컴파일 명령어가 정상적으로 실행되었다.  $ ls: 컴파일 결과, 실행 파일 proxy\_cache가 생성된 것을 확인했다. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2. Operation    먼저 Proxy 2-4 과제와 동일하게 <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html> 을 브라우저 상에 입력하여 MISS 1번을 수행했다.  이후, 같은 주소에 대해 브라우저에서 빠르게 5회 새로고침을 수행하여 자식 프로세스들이 동시에 logfile.txt에 접근하려는 상황을 유도하였다.  또한 write\_log\_sync() 함수 내 Critical Section에 sleep(1)을 삽입함으로써, 여러 자식 프로세스들이 sermaphore에 접근하여 순차적으로 동기화된 로그 출력이 터미널에 나타나는 모습을 확인할 수 있었다.  다음은 pid별 세부적인 동작 해석이다.   |  |  |  | | --- | --- | --- | | 순서 | pid | 동작 | | 1 | 16561 | MISS -> semaphore wait -> 진입 -> exit | | 2 | 16571 | wait (동시 접근)-> 16561 종료 후 진입 (MISS는 빠르게 진행하지 않음) | | 3 | 16572 | wait (동시 접근) | | 4 | 16573 | wait (동시 접근) | | 5 | 16574 | wait (동시 접근) | | 6 | 16575 | wait (동시 접근) | | 7 | 16571 | exit | | 8 | 16572 | 진입 -> exit | | 9 | 16573 | 진입 -> exit | | 10 | 16574 | 진입 -> exit | | 11 | 16575 | 진입 -> exit | | 12 | 16557 | Ctrl+C(SIGINT) 직후 fork되어있던 자식 -> wait -> 진입 -> exit |   첫 pid # 16561은 MISS 로깅이 완료될 때까지 기다리며 다른 브라우저 요청을 보내지 않았고, pid # 16571~16575 5개의 자식 프로세스는 새로고침을 빠르게 5번 진행함으로써 동시 접근 및 wait하는 시나리오가 진행됐다. 이후 순서대로 진입 및 로그 기록 후 exit를 수행했다. 마지막 16557은 SIGINT 직후에도 안전하게 종료 로그를 기록하고 exit 한다.  한 번에 하나의 자식만 logfile.txt에 접근하고 있으며 sleep(1) 덕분에 대기열이 만들어지고, 이 순서대로 진입하는 모습이 잘 보인다.    MISS일 때 browser에 web server가 보내온 응답과 같은 화면이 HIT일 때도 잘 출력된다. |

|  |
| --- |
| 3.logfile    로그파일을 확인한 결과, 처음 요청에 대해 1건의 MISS가 발생했고, 이후 같은 URL에 대해 5건의 HIT 로그가 기록되었다.  이 외에도 브라우저의 자동 요청(favicon)으로 인해 총 2건의 추가적인 자식 프로세스가 생성되어 로그는 작성되지 않았지만 캐시파일이 생성되었다.  최종 종료 로그에는 총 실행 시간과 #sub process: 8로 전체 자식 프로세스 수가 정확히 기록되었으며, 이는 수동 요청 1회 + 새로고침 5회 + 자동 요청 2회가 반영된 값이다.  터미널에 출력된 semaphore 관련 로그는 logfile.txt에 접근하려는 자식 프로세스들의 PID에 한정되므로, 실제 종료 로그에 기록된 전체 자식 프로세스 수와는 차이가 발생한다. 이는 자동 요청(favicon 등)처럼 로그 기록 조건을 만족하지 않는 요청은 HIT/MISS를 기록하지 않아 logfile.txt에 접근하지 않기 때문에 semaphore 동기화 구간에 진입하지 않고, 그에 따라 터미널에도 출력되지 않는 것이다. |

|  |
| --- |
| 4. cache directory    $ ls -R ~/cache: ls 명령어를 –R 옵션을 통해 재귀적으로 실행하여 ~/cache 디렉토리와 그 하위 디렉토리까지 출력하여 캐시 디렉토리와 파일이 생성됨을 확인했다.    $ tree ~/cache/: ~/cache 구조 확인 결과, SHA1 해시된 url의 앞 3글자를 이름으로 하여 디렉토리가 생성되었다. 그 디렉토리의 하위에는 나머지 37글자 이름으로 파일이 생성되었다.  17f/... 캐시는 직접 입력한 <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html> url에 대한 SHA1 해시값에 해당하는 캐시 파일이다.  38d/... 캐시는 자동으로 요청된 <http://gaia.cs.umass.edu/favicon.ico> url에 해당하는 캐시 파일로,  로그에는 기록되지 않지만 실제로는 origin 서버로 요청이 나가고 응답이 캐시에 저장되었다.  이처럼 로그는 수동 요청만 기록, 캐시는 모든 요청 저장 구조가 정확히 반영되었음을 확인할 수 있다.        두 캐시 파일을 열어 MISS일 때 web server로부터의 response가 저장된 것을 확인했다. |

고찰

이번 Proxy 3-1 과제는 Proxy 2-4 과제를 기반으로 하되, 로그파일 접근에 대한 동기화를 구현하는 것이 핵심 목표였다. 기존 2-4 코드 구조는 그대로 유지하되, 추가 요구사항인 동시 접근 제어를 위해 세마포어를 활용한 동기화 기능만을 최소한으로 삽입하는 방향으로 구현하였다. 따라서 터미널 상의 출력문은 모두 과감히 제거하고, 시뮬레이션 결과와 분석은 보고서 상의 시나리오 설명을 통해 명확히 전달할 수 있도록 구성하였다.

특히 세마포어의 동시 접근 시나리오를 유도하기 위해 브라우저에서 매우 빠르게 새로고침을 반복하는 방식으로 요청이 겹치도록 실험 환경을 구성하였다. Critical Section 내에 sleep(1)을 삽입함으로써, 하나의 자식 프로세스가 로그파일에 접근 중일 때 다른 프로세스들이 대기하는 상황을 자연스럽게 연출할 수 있었고, 실제 터미널 출력에서도 세마포어 대기 → 진입 → 해제 순서를 확인할 수 있었다.

초기 내가 한 구현은 POSIX 세마포어 기반이었지만, 과제 요구사항을 다시 확인하던 중 "semkey 값은 포트 번호와 같게 할 것"이라는 조건을 발견했다. 이로 인해 POSIX에서 System V 세마포어 방식으로 전환하게 되었고, semget, semop, semctl 등을 사용하여 구현을 수정하였다.

여기서 semkey는 System V IPC(Inter-Process Communication)에서 사용하는 세마포어 식별자 키 값으로, 동일한 키 값을 사용하는 프로세스끼리 세마포어를 공유할 수 있도록 해준다. 과제에서는 이 키를 포트 번호로 고정함으로써 실행 환경 간 일관성을 유지하도록 요구하였다.

전반적으로 세마포어만 추가하는 간단한 작업이었지만, 기존 구조를 손상시키지 않고 안전하게 삽입해야 했기 때문에 코드 흐름을 면밀히 파악하고 삽입 위치를 신중히 고려해야 했다. 이를 통해 세마포어의 P연산, V연산 개념과 System V 방식 구현법을 직접 체득할 수 있었고, 프로세스 동기화의 실제 동작 원리를 명확히 이해할 수 있는 계기가 되었다.

Reference

시스템프로그래밍 이론 및 실습 자료 참고하였습니다.