Assignment #1: Finding Sequence
System Programming

202111322 오색빛

제출일: 2022.10.30.

Design

- 프로그램 실행 시, 찾아야할 substring(str)과 fork 할 프로세스의 개수(i)를 인자로 전달하면, 미리 지정된 input 배열에서 모든 substring 을 찾아 각 sequence 의 시작 index 를 모두 출력한다.
- 자식 프로세스를 i 개 만들어 각 프로세스가 input 배열을 비슷한 양으로 나눠 병렬처리한다.
- overlap area 에 해당하는 부분에 있는 sequence 도 찾아 출력할 수 있도록 처리한다.
- 각 자식 프로세스는 탐색하면서 시퀀스를 찾으면 pipe 에 시작 index 를 write 한다.
- 각 자식 프로세스는 자신에게 할당된 개수를 탐색하면 종료한다.
- 부모 프로세스는 자식 프로세스가 모두 종료하면 pipe 에서 모두 read 한다.
- pipe 로 읽은 sequence 의 시작 index 를 정렬하여 출력한다.

Implementation

1. Valuable Description

Type	Identifier	Description
const char*	str	찾을 substring (argv[1])
const int	pcount	자식 프로세스의 개수 (argv[2])
pid_t *	pid	자식 프로세스 pid 저장 배열
int *	N	각 프로세스가 탐색해야할 문자의 개수 배열
int *	pipefd	pipe file descriptors (pipefd[0]: read end, pipefd[1]: write end)
int *	foundIndex	부모 프로세스에서 자식 프로세스들이 찾은 sequence 의 시작 index 를 저장하는 배열
int	count	탐색의 시작부분을 계산하기 위한 변수
int	buf	pipe 에서 read 할 때 사용되는 버퍼
int	fcount	찾은 sequence 의 개수
int	min	선택정렬에 사용되는 변수, 최솟값 index

2. Code Description

(1) Main 함수

```
/* error */
if(argc != 3 || str == NULL || pcount  <= 0) {
        printf("argument error\n");
        exit(EXIT_FAILURE);
}

if(pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
}
```

main 함수로 들어온 인자가 3 개가 아닌 경우, 찾으려는 문자열이 없는 경우(NULL), 병렬 프로세스의 개수가 0 보다 작거나 같은 경우 종료하도록 예외처리 했다.
파이프가 생성되지 않은 경우에도 종료하도록 했다.

```
/* calculate N */
for(int i=0; i<pcount; i++) {
        N[i] = MAXS / pcount;
}

if((MAXS%pcount) != 0) {
        for(int i=0; i<(MAXS%pcount); i++) {
                N[i] += 1;
        }
}
```

각 프로세스마다 비슷한 양의 일을 처리하도록 처리할 개수를 담는 배열을 생성했다.

각 프로세스는 우선 input 배열의 크기를 프로세스 개수로 나눈 몫을 나눠 갖고, 만약 나머지가 존재한다면 앞에서부터 1 씩 나눠 갖도록 구현했다.

ex) MAXS = 10, pcount = 3 ➔ N[] = { 4, 3, 3 }

```
/* fork & find sequence */
for(int i=0; i<pcount; i++) {
        count += N[i]; /* update count */
        pid[i] = fork();
        if(pid[i] == -1) {
                perror("fork");
                exit(EXIT_FAILURE);
        }
        else if(pid[i]  == 0) { /* child process */
                close(pipefd[0]);
                findStr(str, count-N[i], N[i], pipefd[1]);
                close(pipefd[1]);
                exit(EXIT_SUCCESS);
        }
}
```

부모 프로세스는 pcount 개수만큼의 자식 프로세스를 생성한다.(fork)

fork 전 count 값을 갱신해서 각 자식 프로세스가 처리해야할 범위를 계산한다.

자식 프로세스는 쓰기 기능만 하면 되므로 사용되지 않는 read end(pipefd[0])은 close 한다.

자식 프로세스는 findStr 함수를 통해 input 에서 substring sequence 를 찾아 pipe 에 write 한다.

처리해야할 범위를 끝내면 write end(pipefd[1])를 close 하고 종료한다.

```
/* read from pipe */
close(pipefd[1]);
wait(NULL);
while(read(pipefd[0], &buf, sizeof(int))>0)
        foundIndex[fcount++] = buf;
close(pipefd[0]);
```

부모 프로세스는 읽기 기능만 하면 되므로 사용하지 않는 write end(pipefd[1])은 close 한다.

자식 프로세스가 모두 종료되면 pipe 에 있는 값을 모두 read 하여 foundIndex 배열에 저장한다.

더 이상 읽을 값이 없을 경우 read end(pipefd[0])를 close 한다.

```
/* sort index */
for(int i=0; i<fcount-1; i++) {
        min = i;
        for(int j=i+1; j<fcount; j++) {
                if(foundIndex[min] > foundIndex[j])
                        min = j;
        }
        if(min != i) {
                int temp = foundIndex[min];
                foundIndex[min] = foundIndex[i];
                foundIndex[i] = temp;
        }
}
```

선택 정렬을 이용해서 foundIndex 를 오름차순으로 정렬했다.

배열을 순차 탐색하면서 최솟값을 찾아 앞부분과 자리를 바꾸어 정렬한다.

```
/* print index */
for(int i=0; i<fcount; i++)
        printf("%d\n", foundIndex[i]);
```

정렬된 foundIndex 를 출력한다.

(2) findStr 함수 (str: 찾을 문자열, s: 시작 위치, n: 탐색 문자 개수, fd: pipe file descriptor)

```
void findStr(const char* str, int s, int n, int fd) {
        int found = -1;
        int end = s+n;

        for(int k=s; k<end; k++) {
                found = k;
                for(int j=0, i=k; j<strlen(str); j++, i++) {
                        if(i >= MAXS) break;
                        if(input[i] == str[j]) {
                                continue;
                        } else {
                                found = -1;
                                break;
                        }
                }
                if(found != -1) {
                        write(fd, &found, sizeof(found)); /* write to pipe */
                }
        }
}
```

(바깥 for 문) 각 자식 프로세스는 input[s]부터 input[end-1]까지 탐색한다. found 에 검색을 시작하는 index(k)를 저장한다. (안쪽 for 문) input[i]가 str[j]과 같을 경우, j 와 i 값을 증가하면서 str 의 글자수만큼 반복을 진행한다. 같지 않은 부분이 나타날 경우, found 값을 -1 로 바꾸고 break 한다. 안쪽 for 문이 끝날 때 found 가 -1 이 아니라면, 즉 substring 을 찾았다면 found 값(sequence 시작 index)을 pipe 에 write 한다.

문자를 검사할 때 i 변수를 이용하여, i는 end 보다 큰 것이 가능하므로 overlap area 부분도 검사할 수 있도록 처리했다. 바깥 for 문이 n 번 반복되면서 문자열이 겹치더라도 올바른 결과값을 찾아낸다. ex) ABABABA 에서 ABA 찾기 ➔ (0, 2, 4)

Function description

Function name	Arguments	Description
main	int argc	메인함수에 전달되는 정보의 개수
	char* argv[]	프로그램을 실행할 때 메인 함수에 전달되는 인자의 문자열들이 저장되는 배열, argv[0]은 프로그램 실행경로로 고정
	Return Value: int	정상적인 종료: 0, 비정상적인 종료: -1
findStr	const char* str	찾으려고 하는 문자열
	int s	input 배열에서 문자열을 찾기 시작하는 위치 index
	int n	process 에 할당된 탐색할 문자의 개수
	int fd	write 할 pipe file descriptor (write end)
	Return Value: void	