유닉스 프로그래밍 및 실습

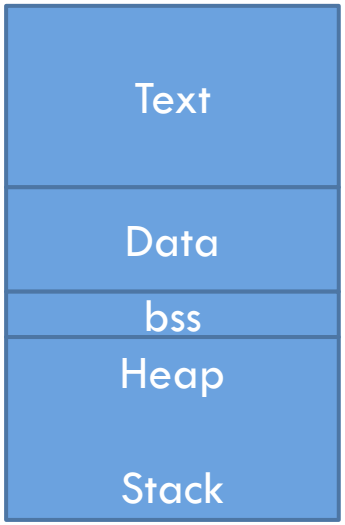
8장. 메모리 관리

1. 프로세스 주소 공간(1)

- 가상메모리(Virtual Memory)
  - 독자적인 선형 주소공간 (0 ~ 최대값)
  - 페이지(page)/페이징(paging)
  - 스왑(swap) 파티션 / 디스크 파일
  - 유효하지 않는 페이지
    - 물리 메모리와 관련을 맺지 않은, 할당되지 않은 주소공간 => 세그먼테이션 폴트
    - 선형적으로 접근하지만 물리 주소를 부여하지 않은 공간이 상당히 많다.
    - 물리 메모리와 관련을 맺지 않은 2차 저장소에 존재하는 페이지를 접근하려는 경우 => 페이지 폴트
  - 물리메모리보다 가상 메모리가 훨씬 크다
- 공유, 쓰기 후 복사
  - 서로 다른 프로세스가 소유한 가상 주소 공간에 존재하는 다중 페이지가 단일 물리 페이지(프레임)에 사상되는 경우
  - 쓰기 가능한 공유 페이지에 쓰려고 하는 경우
    - 모든 프로세스가 다른 프로세스가 쓰기를 수행한 결과를 바로 볼 수 있는 경우
    - 새로운 페이지 복사본을 만들어 새로운 페이지에 쓰기를 허용 (copy-on-write)

1. 프로세스 주소 공간(2)

- 메모리 영역
  - 텍스트 세그먼트
    - 프로그램 코드, 문자열 상수, 상수 변수, 기타 읽기 전용 자료 포함
    - 목적 파일로부터 직접 사상
  - 스택
    - 실행 스택
    - 깊이가 늘어나고 감소함에 따라 동적으로 커지고 줄어듬
    - 지역변수, 함수 반환 자료 포함
  - 데이터 세그먼트 / 힙
    - 프로세스의 동적 메모리 포함
    - 쓰기가 가능하고 크기를 늘리고 줄일 수 있음
  - bss(block started by symbol) 세그먼트
    - 초기화하지 않은 전역변수 포함
    - C 표준에 따라 모두 0인 특수 값을 포함
  - 리눅스에서는 변수를 두가지 방법으로 최적화
    - bss 세그먼트는 초기화하지 않은 자료를 위한 전용공간으로 할당(링커가 목적파일에 특수 값을 저장하지 않으므로 바이너리 크기가 줄어듬)
    - 세그먼트를 메모리에 올릴 때 쓰기후 복사 기법으로 0으로 가득 찬 페이지에 사상하여 기본 값 설정



2. 동적 메모리 할당하기 (1)

- 동적 메모리를 대상으로 할당, 사용, 반환함
- 컴파일 시점이 아닌, 실행 중에 할당
- 동적 메모리를 지원하는 C 변수는 없지만 동적 메모리를 얻는 메커니즘 지원
- 고전적인 C 인터페이스 malloc()

```
#include <stdlib.h>

void * malloc(size_t size);
```

  - 성공하면 size 만큼 메모리 할당하고 포인터 반환 (메모리 내용은 정의되지 않은 상태)
  - 실패하면 NULL
- 예제 코드
- C 에서는 자동으로 반환하는 타입으로 변환하지만, C++ 사용자는 타입 변환이 필요함
  - 예제코드 비교
- NULL이 반환될 수 있으므로, 항상 오류조건 확인할 것!
- 예제 코드

2. 동적 메모리 할당하기 (2)

- 배열 할당하기

```
#include <stdlib.h>

void * calloc(size_t nr, size_t size);
```

  - 성공하면 크기가 size인 배열 원소를 nr만큼 담을 수 있는 메모리 블록을 가리키는 포인터 반환 (단 메모리 영역을 모두 0으로 초기화)
  - 예제코드 비교 (malloc과 calloc)
  - memset을 이용하는 것보다 calloc이 더 효율적인 것은 커널이 이미 0으로 채워진 메모리를 제공하기 때문
  - 예제코드 (메모리 할당 후 0으로 채우기)

2. 동적 메모리 할당하기 (3)

- 할당 크기 조정

```
#include <stdlib.h>

void * realloc(void *ptr, size_t size);
```

  - 성공하면 ptr로 가리키는 공간의 크기를 size로 조정
  - 이때 동일한 ptr이 아닐 수도 있음 (크기를 키우는 경우 새로 공간을 할당하고 내용을 모두 복사하는 일이 벌어질 수도 있음)
  - size가 0이면 free와 동일
  - 예제코드 (calloc 후 realloc)

2. 동적 메모리 할당하기 (4)

- 동적 메모리 해제하기

```
#include <stdlib.h>

void free(void *ptr);
```

  - 이때 ptr은 반드시 전에 malloc, calloc, realloc이 반환한 값이어야 함 (절반만 해제할 수 없음)
  - ptr이 NULL이어도 상관없음 – 확인하는 것은 중복
  - 예제 코드
  - 메모리 누수 – 동적 할당 후 free 수행을 잊었을 때
  - free 이후 접근 – 어느 시점까지는 내용이 남아있지만, 그 기간을 알 수 없음
  - C 언어는 메모리 관리 책임이 프로그래머에게 있다.
    - 연결이 끊어진 포인터, NULL은 아니지만 유효하지 않은 블록을 가리키는 포인터에 특히 주의를 기울여야
    - Electric fence, valgrind 같은 유틸리티 고려

2. 동적 메모리 할당하기 (5)

- 정렬(align)
  - 자료 정렬 : 하드웨어가 다루는 주소와 하드웨어가 다루는 메모리 블록 사이에 존재하는 관계
    - 32비트 변수는 4의 배수인 메모리 주소에 위치하는 경우 자연스럽게 정렬된 상태라고 본다.
  - 규칙은 하드웨어마다 다르다
  - 정렬되지 않은 자료 접근이 가능한 구조에서도 정렬을 고려하는 것이 효율적
- 정렬된 메모리 할당하기
  - POSIX는 malloc(), calloc(), realloc()이 적절히 정렬되어야 한다고 정의
  - 리눅스의 경우 32비트 시스템에서는 8바이트 경계에, 64비트 시스템에서는 16바이트 경계에 정렬된 메모리를 반환
  - 페이지처럼 더 큰 경계로 정렬된 동적 메모리 요구시
    - posix_memalign() 함수 이용
    - free()로 해제
    - 예제 코드
  - 예전 인터페이스
    - BSD, SunOS

2. 동적 메모리 할당하기 (6)

- 다른 정렬 고려사항
  - 표준 타입과 동적 메모리 할당이 요구하는 자연스러운 정렬을 벗어난 확장 형태에서 특히 중요
  - 비표준 타입
    - 구조체 정렬 요구사항은 가장 큰 구성 요소 타입에 따른다.
    - 구조체에는 채워넣기가 필요한 경우가 있다.
    - 공용체의 경우 가장 큰 타입을 따른다.
    - 배열의 경우 기본 타입을 따른다.
- 포인터로 작업하기
  - 예제 코드(정렬을 지키지 못한 코드) 이해

3. 자료 세그먼트 관리하기

- UNIX는 역사적으로 자료 세그먼트를 직접 다루기 위한 인터페이스를 제공함
  - 대부분 직접 활용하지 않은 것은 malloc() 같은 함수들이 좀더 사용하기 쉽고 강력하기 때문

```
#include <unistd.h>

int brk(void *end);
void *sbrk(intptr_t increment);
```

  - 두 함수의 이름은 힙과 스택이 동일 세그먼트에 존재했던 전통적 유닉스 시스템에 따라 명명
  - brk() : 차단 지점을 end가 명세한 주소로 설정
  - sbrk() : increment만큼 자료 세그먼트 끝을 증가

과제

1. 호텔에서 한 층의 객실 갯수와 층 갯수를 받아들여 고객 명단을 관리할 배열 생성 후 입실, 퇴실, 숙박 중인 고객명단 출력 기능 수행
2. 호텔 구조(층별 객실 수와 층 수) 변경시 범위를 벗어난 데이터들만 빈 방으로 이동하도록 기능 구현 (빈 방이 부족한 경우 오류 메시지 출력)

- 제출기한 : 11월 13일 자정