컴퓨터학부 20221867 이채은

**1. 개요**

- xv6 운영체제의 메모리 관리 시스템을 이해하고 확장하는 것을 목표로 한다. 구체적으로는 메모리의 지연 할당 및 해제 기능을 구현하는 ssusbrk() 시스템 콜과 메모리 상태를 확인하는 memstat() 시스템 콜을 구현한다. ssusbrk() 시스템 콜에서 메모리 할당 시에는 처음에 가상 메모리만 할당하고, 실제 물리 메모리는 해당 페이지에 접근할 때 페이지 폴트를 통해 할당된다. 이는 메모리를 효율적으로 사용하기 위함이다. 메모리 해제의 경우, 지정된 tick만큼 지연된 후에 해제가 이루어지며, 이전 해제 요청이 처리되지 않은 상태에서 새로운 해제 요청이 들어오면 두 요청이 병합되고 tick은 새로운 요청을 기준으로 초기화된다. 이러한 기능의 정상 작동 여부를 확인하기 위해 memstat() 시스템 콜을 구현한다. 현재 프로세스의 가상 메모리 페이지와 물리 메모리 페이지의 개수, 그리고 페이지 디렉토리 엔트리(PDE)와 페이지 테이블 엔트리(PTE)의 값을 확인할 수 있다. 이를 통해 메모리 할당과 해제가 올바르게 이루어지는지 검증할 수 있다. 이 과제에서는 운영체제의 메모리 관리 방식, 특히 페이지 테이블의 구조와 동작 방식, 가상 메모리와 물리 메모리의 매핑, 그리고 시스템 콜을 통한 메모리 관리 기법에 대해 배울 수 있는 시간이 주어진다.

**2. 상세 설계 & 3. 실행 결과 & 4. 주석 달린 소스코드**

**- 1-1. ssusbrk() 시스템 콜 구현**

- **(기능 1)** 첫 번째 매개변수가 양수인 경우 입력된 페이지의 크기만큼의 메모리를 지연 할당함, (기능 2) 부분도 한번에 주석으로 설명

|  |
| --- |
| sysproc.c |
| **. . .** 기존 코드와 동일  // 구조체 초기화 - 메모리 해제 요청 정보를 저장할 구조체  struct dealloc\_info pending\_dealloc = {0, 0, 0, 0, 0}; // (기능 2) 구현을 위해 삽입, 보고서에 첨부한 proc.h 참고  // 시스템 호출 구현: ssusbrk  int  sys\_ssusbrk(void)  {  int n; // 첫 번째 매개변수를 저장할 변수  int ticks\_delay; // 두 번째 매개변수를 저장할 변수  struct rtcdate t; // 날짜와 시간 정보를 저장할 구조체  struct proc \*curproc = myproc(); // 현재 프로세스 정보 가져오기    // 매개변수 유효성 확인  if(argint(0, &n) < 0 || argint(1, &ticks\_delay) < 0)  return -1;  // 첫 번째 매개변수(n)가 유효한지 확인  if(n == 0 || (n > 0 && n % PGSIZE != 0)) // 0이거나 페이지 크기의 배수가 아닌 경우  return -1;    // 메모리 할당 요청 처리  if(n > 0) {  uint old\_sz = curproc->sz; // 현재 프로세스의 메모리 크기 저장  uint new\_sz = old\_sz + n; // 새로운 메모리 크기 계산    if(new\_sz >= KERNBASE) // 커널 메모리 영역 침범 방지  return -1;    curproc->sz = new\_sz; // 프로세스 메모리 크기 갱신  return old\_sz; // 기존 메모리 크기 반환    } else { // 메모리 해제 요청 처리  if(ticks\_delay <= 0) // 유효하지 않은 딜레이 값이면 반환  return -1;    // 매개변수 n을 양수로 변경 (해제 크기 계산)  n = -n;  if(n % PGSIZE != 0 || n > curproc->sz) // 페이지 크기의 배수가 아니거나 할당된 크기를 초과한 경우  return -1;    // 현재 날짜 및 시간 정보 출력  cmostime(&t);  cprintf("Memory deallocation request(%d): %d-%d-%d %d:%d:%d\n",  ticks\_delay, t.year, t.month, t.day, t.hour, t.minute, t.second);    // tickslock을 사용해 병렬 처리 방지  acquire(&tickslock);  if(pending\_dealloc.active) { // 이전 요청이 이미 활성화된 경우  // 이전 요청과 새로운 요청 병합  pending\_dealloc.size += n; // 해제할 크기를 합산  pending\_dealloc.ticks\_delay = ticks\_delay; // 새로운 딜레이 값으로 갱신  pending\_dealloc.request\_ticks = ticks; // 요청 시점을 현재 시점으로 갱신  } else { // 새로운 요청인 경우  pending\_dealloc.addr = curproc->sz - n; // 해제 시작 주소 설정  pending\_dealloc.size = n; // 해제할 크기 설정  pending\_dealloc.ticks\_delay = ticks\_delay; // 딜레이 값 설정  pending\_dealloc.request\_ticks = ticks; // 요청 시점 설정  pending\_dealloc.active = 1; // 요청 활성화  }  release(&tickslock); // tickslock 해제    return curproc->sz - n; // 새로운 메모리 크기 반환  }  } |

✓ 첫 번째 매개변수는 할당할 메모리의 크기(0은 안됨)이며, 페이지 크기의 배수임

- 유효한 값을 입력받고 가상 메모리를 성공적으로 할당했다면 할당한 메모리의 주소를 반환

if(argint(0, &n) < 0 || argint(1, &ticks\_delay) < 0)

return -1;

if(n == 0 || (n > 0 && n % PGSIZE != 0))

return -1;

🡪 이 경우 두 번째 매개변수의 값이 무엇이든 무시됨

🡪 유효하지 않은 값을 입력받았을 때는 -1을 반환

✓ ssusbrk() 시스템 콜이 호출됐을 때 가상 메모리만 할당하고 물리 메모리는 할당하지 않음 🡪 ssusbrk() 주석 참고

- 할당된 가상 메모리에 사용자가 접근할 때 해당 가사 메모리에 대응하는 물리 메모리가 존재하지 않을 경우 특정 trap이 발생하며, 이때 물리 메모리를 페이지 단위로 할당하도록 구현

-둘 이상의 가상 메모리 페이지가 할당된 경우 접근된 페이지에 대해서만 물리 메모리 페이지를 할당함, (기능 2) 부분도 한번에 주석으로 설명

|  |
| --- |
| tarp.c |
| **. . .** 기존 코드와 동일  #include "date.h"  **. . .**  extern int sys\_memstat(void);  extern struct dealloc\_info pending\_dealloc;  void  trap(struct trapframe \*tf)  {  // 페이지 폴트 예외 처리  if(tf->trapno == T\_PGFLT) {  uint va = rcr2(); // 잘못된 메모리 접근 주소 확인  struct proc \*curproc = myproc(); // 현재 실행 중인 프로세스 가져오기    // 잘못된 접근 주소가 프로세스의 메모리 범위 내에 있는지 확인  if(va < curproc->sz && va >= PGSIZE) {  pde\_t \*pde = &curproc->pgdir[PDX(va)]; // 페이지 디렉토리 엔트리 가져오기    // 페이지 디렉토리에 엔트리가 존재하지 않는 경우 처리  if(!(\*pde & PTE\_P)) {  char \*new\_pt = kalloc(); // 새로운 페이지 테이블 할당  if(new\_pt == 0) { // 메모리 부족 시 오류 처리  cprintf("out of memory in page table allocation\n");  curproc->killed = 1;  return;  }  memset(new\_pt, 0, PGSIZE); // 새로 할당한 페이지 테이블 초기화  \*pde = V2P(new\_pt) | PTE\_P | PTE\_W | PTE\_U; // 페이지 디렉토리 업데이트  }    // 페이지 테이블 엔트리 가져오기  pte\_t \*pgtab = (pte\_t\*)P2V(PTE\_ADDR(\*pde));  pte\_t \*pte = &pgtab[PTX(va)];    // 페이지 테이블에 엔트리가 존재하지 않는 경우 처리  if(!(\*pte & PTE\_P)) {  char \*mem = kalloc(); // 새로운 페이지 할당  if(mem == 0) { // 메모리 부족 시 오류 처리  cprintf("out of memory\n");  curproc->killed = 1;  return;  }  memset(mem, 0, PGSIZE); // 새로 할당한 페이지 초기화  \*pte = V2P(mem) | PTE\_P | PTE\_W | PTE\_U; // 페이지 테이블 업데이트  }  return; // 페이지 폴트 처리 완료 후 반환  }  }  // 시스템 호출 처리  if(tf->trapno == T\_SYSCALL){  if(myproc()->killed) // 프로세스가 종료 상태이면 종료  exit();  myproc()->tf = tf; // 현재 트랩 프레임 저장  syscall(); // 시스템 호출 처리  if(myproc()->killed) // 처리 중 프로세스가 종료 상태가 되면 종료  exit();  return;  }  switch(tf->trapno){  case T\_IRQ0 + IRQ\_TIMER: // 타이머 인터럽트 처리  if(cpuid() == 0){  acquire(&tickslock); // 타이머 동기화용 락 획득  ticks++; // 전역 타이머 값 증가  struct proc \*curproc = myproc(); // 현재 실행 중인 프로세스 가져오기  if(pending\_dealloc.active) { // 지연된 메모리 해제 요청이 활성화된 경우  if(ticks - pending\_dealloc.request\_ticks >= pending\_dealloc.ticks\_delay) { // 요청 대기 시간이 지난 경우  // 메모리 해제 시작  uint new\_sz = curproc->sz - pending\_dealloc.size; // 새로운 프로세스 메모리 크기 계산    release(&tickslock); // 락 해제  if(deallocuvm(curproc->pgdir, curproc->sz, new\_sz) == 0) { // 물리 메모리 해제  cprintf("deallocation failed\n"); // 해제 실패 시 메시지 출력  acquire(&tickslock);  return;  }  curproc->sz = new\_sz; // 프로세스 메모리 크기 업데이트    // 메모리 해제 완료 시간 출력  struct rtcdate t;  cmostime(&t);  cprintf("Memory deallocation execute: %d-%d-%d %d:%d:%d\n",  t.year, t.month, t.day, t.hour, t.minute, t.second);    // 메모리 상태 확인  sys\_memstat();    acquire(&tickslock); // 락 재획득  pending\_dealloc.active = 0; // 요청 비활성화  pending\_dealloc.size = 0; // 요청 크기 초기화  curproc->killed = 1; // 프로세스 종료 설정  }  }    wakeup(&ticks); // 대기 중인 프로세스 깨우기  release(&tickslock); // 락 해제  }  lapiceoi(); // 인터럽트 종료 처리  break;  **. . .**  // 실행 중인 프로세스가 RUNNING 상태이면서 타이머 인터럽트가 발생한 경우  if(myproc() && myproc()->state == RUNNING &&  tf->trapno == T\_IRQ0+IRQ\_TIMER) {  if(!pending\_dealloc.active) { // 지연된 메모리 해제가 없을 경우  yield(); // CPU를 다른 프로세스에 양보  }  } |

- **(기능 2)** 첫 번째 매개변수가 음수인 경우, 입력된 페이지의 크기만큼의 메모리를 두 번째 매개변수의 tick만큼 지연시킨 후 해제함

- (기능 2) 구현을 위해 메모리 해제를 위한 struct인 dealloc\_info 생성

|  |
| --- |
| proc.h |
| A screen shot of a computer code  Description automatically generated |

✓ 첫 번째 매개변수는 해제할 메모리의 크기(0은 안됨)이며, 페이지 크기의 배수이고, 두 번째 매개변수는 메모리 해제 시 지연시킬 tick의 값이며, 양수만 가능함

if(ticks\_delay <= 0) // 유효하지 않은 딜레이 값이면 반환

return -1;

// 매개변수 n을 양수로 변경 (해제 크기 계산)

n = -n;

if(n % PGSIZE != 0 || n > curproc->sz) // 페이지 크기의 배수가 아니거나 할당된 크기를 초과한 경우

return -1;

- 유효한 값을 입력받고, 가상 메모리를 성공적으로 해제했다면 해제한 메모리의 주소를 반환

🡪 유효하지 않은 값을 입력받았을 때는 -1을 반환

✓ ssusbrk() 시스템 콜이 호출되고 입력받은 tick값이 지난 후 메모리 페이지 해제 작업이 수행됨

- 이전에 요청된 메모리 해제가 수행되지 않은 시점에 다시 ssusbrk()가 호출된 경우 이전에 해제 요청한 페이지에 새로 요청된 페이지가 더해지고, tick값은 뒤에 호출된 ssusbrk()의 값을 기준으로 초기화됨

if(pending\_dealloc.active) { // 이전 요청이 이미 활성화된 경우

// 이전 요청과 새로운 요청 병합

pending\_dealloc.size += n; // 해제할 크기를 합산

pending\_dealloc.ticks\_delay = ticks\_delay; // 새로운 딜레이 값으로 갱신

pending\_dealloc.request\_ticks = ticks; // 요청 시점을 현재 시점으로 갱신

} else { // 새로운 요청인 경우

// 안의 내용은 위에 첨부된 ssusbrk() 구현 부분 참고

}

🡪 시스템 콜이 호출된 시점과 해제 작업이 수행된 시점의 날짜 및 시간 정보를 화면에 출력하고, 해제된 메모리의 상태를 확인하기 위해 memstat()을 호출해야 함 🡪 trap() 구현 코드 참고, sys\_memstat();

🡪 테스트 프로그램을 수행하기 위해 메모리 해제가 이뤄진 후 프로세스를 종료하도록 구현해야 함

**- 1-2. memstat() 시스템 콜 구현**

- (기능 1) 호출한 프로세스의 가상 메모리 페이지와 물리 메모리 페이지의 개수를 화면에 출력

- (기능 2) 현재 물리 메모리로 할당되어 있는 페이지 디렉토리 엔트리(PDE)와 페이지 테이블 엔트리(PTE)의

값을 화면에 출력

|  |
| --- |
| sysproc.c |
| **. . .** 기존 코드와 동일  int  sys\_memstat(void)  {  struct proc \*curproc = myproc(); // 현재 실행 중인 프로세스 가져오기  pde\_t \*pgdir = curproc->pgdir; // 프로세스의 페이지 디렉토리 가져오기  uint vp = 0, pp = 0; // 가상 페이지 수(vp)와 실제 물리 페이지 수(pp) 초기화  pte\_t \*pgtab; // 페이지 테이블 포인터  uint va; // 가상 주소  // 프로세스 크기를 기준으로 가상 페이지 수 계산  vp = PGROUNDUP(curproc->sz) / PGSIZE;  cprintf("vp: %d, ", vp); // 가상 페이지 수 출력  // 프로세스의 가상 주소 공간을 순회하며 물리 페이지 수 계산  for(va = 0; va < curproc->sz; va += PGSIZE) {  pde\_t \*pde = &pgdir[PDX(va)]; // 가상 주소에 해당하는 페이지 디렉토리 엔트리 가져오기  if(\*pde & PTE\_P) { // 페이지 디렉토리 엔트리가 존재하면  pgtab = (pte\_t\*)P2V(PTE\_ADDR(\*pde)); // 페이지 테이블의 시작 주소 가져오기  pte\_t \*pte = &pgtab[PTX(va)]; // 페이지 테이블 엔트리 가져오기  if(\*pte & PTE\_P) { // 페이지 테이블 엔트리가 존재하면  pp++; // 물리 페이지 수 증가  }  }  }  cprintf("pp: %d\n", pp); // 물리 페이지 수 출력    // 첫 번째 페이지 디렉토리 엔트리(PDE) 출력  pde\_t \*pde = &pgdir[0];  cprintf("PDE - 0x%x\n", \*pde);    // 페이지 테이블 엔트리(PTE) 출력  cprintf("PTE - ");  int first = 1; // 첫 번째 엔트리 출력 여부 플래그  for(va = 0; va < curproc->sz; va += PGSIZE) {  pde = &pgdir[PDX(va)]; // 가상 주소에 해당하는 페이지 디렉토리 엔트리 가져오기  if(\*pde & PTE\_P) { // 페이지 디렉토리 엔트리가 존재하면  pgtab = (pte\_t\*)P2V(PTE\_ADDR(\*pde)); // 페이지 테이블의 시작 주소 가져오기  pte\_t \*pte = &pgtab[PTX(va)]; // 페이지 테이블 엔트리 가져오기  // 페이지 테이블 엔트리가 존재하고 사용자 접근 가능한 페이지만 출력  if((\*pte & PTE\_P) && (\*pte & PTE\_U)) {  if(!first) // 첫 번째 출력이 아니면 구분자("-") 출력  cprintf(" - ");  cprintf("0x%x", \*pte); // 페이지 테이블 엔트리 출력  first = 0; // 첫 번째 출력 여부 플래그 해제  }  }  }  cprintf("\n"); // 출력 줄바꿈    return 0; // 함수 종료  } |

✓ memstat() 호출 결과

|  |
| --- |
| ssusbrk\_test1 실행 결과 |
| A screenshot of a computer program  Description automatically generated |

|  |
| --- |
| ssusbrk\_test2 실행 결과 |
| A screenshot of a computer program  Description automatically generated |

|  |
| --- |
| ssusbrk\_test3 실행 결과 |
| A screenshot of a computer program  Description automatically generated |

- 도출된 결과의 값이 맞는지 확인하기 위해 별도로 테스트를 진행 🡪 PTE/PDE 값은 실제 물리 메모리 주소를 포함하기 때문에 컴퓨터마다, 실행할 때마다 다르게 나올 수 있기 때문

중요한 것은:

1. 페이지 크기의 배수로 메모리가 할당

2. flag 비트가 올바르게 설정

3. vp와 pp의 개수가 정확

4. 메모리 해제가 제대로 동작

- 2, 3, 4는 출력 결과를 통해서 정상적으로 동작하는 것을 확인했고, PTE와 PDE의 안에 저장된 값을 바탕으로 메모리 주소 자체의 값이 페이지 크기의 배수가 맞는지 확인하는 코드를 별도로 만들어 테스트

A screenshot of a computer code

Description automatically generated

**+ 수정한 파일들**

|  |
| --- |
| defs.h |
| **. . .** 기존 코드와 동일  // sysproc.c  int sys\_memstat(void); |

|  |
| --- |
| usys.S |
| **. . .** 기존 코드와 동일  SYSCALL(ssusbrk)  SYSCALL(memstat) |

|  |
| --- |
| user.h |
| // system calls  **. . .** 기존 코드와 동일  int ssusbrk(int, int);  int memstat(void); |

|  |
| --- |
| syscall.h |
| **. . .** 기존 코드와 동일  #define SYS\_ssusbrk 22  #define SYS\_memstat 23 |

|  |
| --- |
| syscall.c |
| **. . .** 기존 코드와 동일  extern int sys\_ssusbrk(void);  extern int sys\_memstat(void);  **. . .**  static int (\*syscalls[])(void) = {  **. . .** 기존 코드와 동일  [SYS\_ssusbrk] sys\_ssusbrk,  [SYS\_memstat] sys\_memstat,  }; |

|  |
| --- |
| Makefile |
| UPROGS=\  ... 기존 코드와 동일  \_ssusbrk\_test1\  \_ssusbrk\_test2\  \_ssusbrk\_test3\  **. . .**  EXTRA=\  ... 기존 코드와 동일  ssusbrk\_test1.c ssusbrk\_test2.c ssusbrk\_test3.c\ |