컴퓨터학부 20211527 박창현

**1. 개요 :**

xv6에 소프트웨어 페이지 워커(sw\_vtop), 물리 프레임 사용 현황 계측(PF 테이블·memdump), 역페이지 테이블(IPT)·소프트 TLB(STLB)를 확장·구현하는 것이다.  
sw\_vtop은 커널이 PDE/PTE를 직접 파싱해 VA→PA를 계산하며, 사용자 도구 vtop으로 물리주소와 PTE 플래그를 조회한다.  
PF 테이블은 모든 물리 프레임의 할당 여부·소유 pid·할당 시각을 추적하고 memdump로 덤프하며, memstress로 부하 상황에서 동작을 검증한다.  
IPT+STLB는 PFN→(pid, va, flags) 역참조와 변환 캐시를 제공하고, 페이지 테이블 변화와 항상 일관성을 유지하도록 갱신한다. 커널 페이지는 추적하지 않으며 사용자 페이지(PTE\_U)만 관리한다.

**2. 상세설계 :**

(2-1) <시스템 콜 설계>

(sys\_dump\_physmem\_info) : 물리 프레임 사용 정보 스냅샷

int dump\_physmem\_info(struct physframe\_info \*u\_buf, int max); (사용자 api)

인자/검증  
 - u\_buf는 유저 가상 주소여야 하며, max \* sizeof(struct physframe\_info) 만큼 쓸 수 있어야 한다.  
 - max > 0 필수.

동작 시퀀스  
 1. 인자 검사 → 길이 산정(n=min(PFNNUM,max))  
 2. 락(pf\_lock/ kmem.lock) 획득  
 3. PF 테이블 스냅샷 생성  
 4. copyout로 유저 버퍼에 전달  
 5. 성공 시 n 리턴

락/동시성  
 - PF 테이블 갱신(kalloc/kfree)과 경합하므로 락으로 보호.  
 - 스냅샷 동안 짧게 잡는 것이 원칙.

반환값  
 - (>=0) : 복사한 엔트리 수  
 - (<0) : 실패

에러 케이스  
 - u\_buf 접근 불가(EFAULT) → -1  
 - max <= 0 또는 내부 오류 → -1

(sys\_vtop) : 현재 프로세스 VA → PA 변환 (소프트 TLB)

int vtop(void \*va, uint \*pa\_out, uint \*flags\_out); (사용자 api)

인자/검증  
 - va: 임의의 유저 VA(페이지 경계 아닐 수 있음)  
 - u\_pa, u\_flags는 유저 쓰기 가능 포인터여야 함

동작 시퀀스  
 1. (pid, va\_page)로 STLB 조회 → hit 시 즉시 완료  
 2. miss면 sw\_vtop(pgdir, va, &pa, &flags)로 PDE/PTE 직접 파싱  
 3. 성공 시 STLB에 fill(miss++)  
 4. pa/flags를 copyout

락/동시성  
 - STLB 접근 시 stlblk 잠금  
 - sw\_vtop은 읽기만 함(락 불필요)

반환값-> 0: 성공, -1: 변환 실패(미매핑/권한 불일치/유저 포인터 오류)

에러 케이스  
 - PDE/PTE 불참 → -1  
 - copyout 실패 → -1

(sys\_phys2virt) 역페이지 질의(IPT) : 주어진 물리 페이지(페이지 정렬된 PA) 를 참조하는 유저 매핑들을 역으로 찾아 (pid, va\_page, flags) 목록으로 돌려준다. 구현은 IPT 해시 테이블을 사용한다.

int phys2virt(uint pa\_page, struct vref \*u\_buf, int max); (사용자 api)

인자/검증  
 - pa\_page는 페이지 정렬된 PA(하위 12비트 0).   
 - u\_buf는 유저 쓰기 가능 포인터  
 - max > 0

동작 시퀀스  
 1. pfn = pa\_page >> 12 계산  
 2. IPT 버킷 체인을 순회해 (pid, va\_page, flags) 수집  
 3. 유저 페이지만(PTE\_U) 남기도록 필터  
 4. 최대 max개를 copyout

락/동시성 : ipt\_query() 내부에서 \*\*iptlk\*\*로 버킷 체인 접근 보호

반환값-> (>=0) : 반환한 엔트리 수(0 가능), (<0): 실패

에러 케이스 : copyout 실패, 인자 오류 → -1

(sys\_tlbstat) 소프트 TLB 통계 조회 : 전역 소프트 TLB의 누적 hit/miss 카운터를 유저로 전달한다.

int tlbstat(uint \*hits\_out, uint \*misses\_out); (사용자 api)

인자/검증 : hits\_out, misses\_out는 유저 쓰기 가능 포인터여야 함

동작 시퀀스  
 1. stlb\_stats(&h,&m)로 원자적 스냅샷  
 2. copyout 두 번

락/동시성 : STLB 카운터 접근은 \*\*stlblk\*\*로 보호

반환값-> 0: 성공, -1: 실패

에러 케이스 : copyout 실패, 인자 오류

(2.2) <커널 변경 사항>

types.h를 공용헤더로 사용 : physframe\_info, vref 구조체 선언

defs.h : (PF 테이블 노출용, 역질의 결과, STLB / IPT API, 내부 도우미) 관련 프로토 타입, 공유 구조체

**syscall.h / syscall.c / usys.S / user.h**  
 - SYS\_dump\_physmem\_info, SYS\_vtop, SYS\_phys2virt, SYS\_tlbstat 번호/디스패치/스텁/프로토타입 추가.

main.c : stlb\_init(), ipt\_init() -> (다른 서브시스템 초기화 뒤, 첫 유저 프로세스 실행 전에 호출)

kalloc.c : 전역테이블 정의, PFN 계산 도우미

vm.c : stlb + ipt + sw\_vtop

(프로세스간 오염방지와 빠른 캐시를 위해 키에 pid포함, stlb\_lookup에서 히트/미스 카운트 집계 -> sys\_tlbstat()으로 노출)

void stlb\_fill(int pid, uint va\_page, uint pa\_page, uint flags) //miss 후 채움. 충돌 시 단순히 교체.

void stlb\_update\_flags(int pid, uint va\_page, uint newflags).// PTE 권한 변경(clearpteu 등) 반영

void stlb\_invalidate\_va(int pid, uint va\_page) // unmap 시 해당 라인 무효화(좀비 히트 방지)

void stlb\_purge\_pid(int pid) // exit/exec 시 이 PID가 쓴 엔트리 전부 제거

IPT (PFN 버킷 체인, 유저 매핑만) : 물리 프레임→(pid, va, flags) 역참조가 빠름. 같은 PFN 다중 매핑 체인보관

void ipt\_insert(uint pfn, int pid, uint va\_page, uint flags) // // kalloc으로 엔트리 할당 → 버킷 앞단에 삽입 (유저페이지만)

void ipt\_remove(int pid, uint va\_page, uint pfn) // 해당 (pid,va\_page,pfn) 엔트리를 체인에서 찾아 제거(kfree)

int ipt\_query(uint pfn, struct vref \*out, int max) // 버킷 체인을 순회해 최대 max개를 out에 채움

void ipt\_update\_flags(int pid, uint va\_page, uint pfn, uint newflags) // 권한 변경 시 해당 엔트리의 flags를 동기화

void ipt\_purge\_pid(int pid) // exit/exec 시 이 PID의 모든 엔트리를 버킷 전역에서 제거

int sw\_vtop(pde\_t \*pgdir, const void \*va, uint \*pa\_out, uint \*flags\_out)

1) PDX/ PTX로 PDE/PTE를 직접 파싱

2) P 비트 확인 (없으면 -1)

3) PTE\_ADDR(pte) | (va & 0xFFF)로 물리주소 조합

4) PTE\_FLAGS(pte)를 그대로 반환

vm.c — 기존 경로에 훅 삽입

- mappages() // 매핑 생성 시점에만 알 수 있는 (pid,va,pfn,flags)를 **IPT에 기록**.

- deallocuvm() // 언매핑시 IPT, STLB 함께 정리

- clearpteu() // 권한 변경 동기화->IPT에도 반영

proc.c / exec.c ->생명주기 purge (메모리 해제가 보통 wait()에서야 끝나지만, **IPT는 즉시 정리,** exec 후 **이전 주소공간의 흔적**을 캐시/표로 남기지 않기 위함)

sysproc.c 추가 시스템 콜 기입(커널 측) -> sys\_dump\_physmem\_info, sys\_vtop, sys\_phys2virt, sys\_tlbstat

(2.3) <호출 순서도>

memstress 페이지 확보(+옵션), 유지, 종료

라인, 텍스트, 영수증, 스크린샷이(가) 표시된 사진

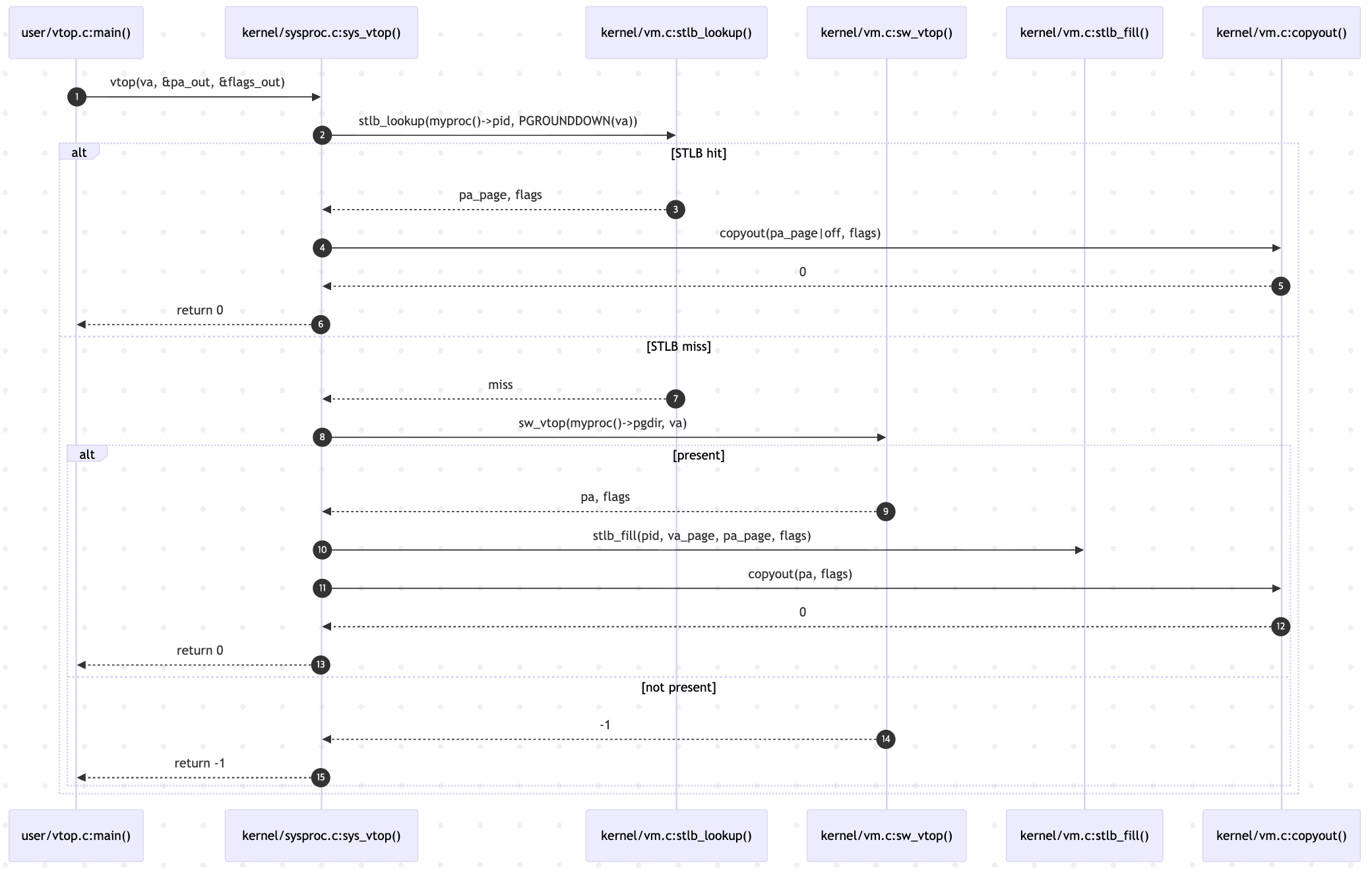
AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

memdump (물리 프레임 회계 스냅샷)

텍스트, 영수증, 스크린샷, 라인이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

vtop (VA->PA 변환, STLB 경유)



pfind (PA\_PAGE 역질의, IPT)

텍스트, 스크린샷, 영수증, 라인이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

tlbstat (STLB 통계 조회)

텍스트, 스크린샷, 라인, 폰트이(가) 표시된 사진

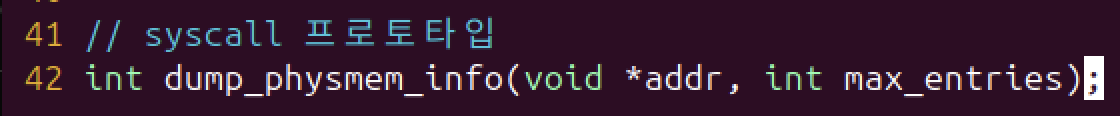
AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

3. 결과 : -(A,B 필수구현부분)  
(1) types.h(공용헤더파일)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

user.h에 추가



(2) usys.S

폰트, 그래픽, 스크린샷, 그래픽 디자인이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.  
3) syscall.h

  
(4) syscall.c



스크린샷, 폰트, 텍스트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.  
sysproc.c

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

kalloc.c

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

defs.h



memstress.c

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

주어진 페이지 수만큼 sbrk()로 가상 메모리를 확장하고(필요 시 각 페이지를 1바이트씩 쓰기), 지정된 시간 동안 유지해 **실제 물리 프레임 할당을 유도**하는 부하 프로그램입니다. 시작 시 파라미터를 에코해 줍니다

**옵션/입출력** -n <pages>: 확보할 페이지 수(기본 10) -t <ticks>: 확보 상태 유지 시간(기본 200 ticks) -w : 각 페이지 첫 바이트에 쓰기(지연할당 구현 시 실제 물리 할당을 보장)

**핵심 동작 흐름  
 1.** 옵션 파싱 → 음수/누락 시 usage() 종료.  **2.** 확보할 바이트 계산(inc = pages\*4096) 후 sbrk(inc)로 힙 확장. 실패 시 에러 출력.  **3.** -w면 페이지당 1바이트 쓰기(오프셋 p\*4096).  **4.** sleep(hold\_ticks)로 유지 후 종료 메시지 출력.

**커널/시스템콜 연계 포인트 :** sbrk() → growproc() → allocuvm() → mappages() → kalloc() 경로로 이어짐

memdump.c

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

커널이 유지하는 **물리 프레임 회계 스냅샷**을 유저 공간으로 받아와, 조건에 맞게 필터링하여 표시합니다. -a가 없으면 **할당된 프레임(alloc=1)** 만, -p PID가 있으면 해당 PID만 출력합니다.

**옵션/입출력 : (**커널에서 복사받는 레코드 타입: struct physframe\_info[frame\_index, allocated, pid, start\_tick]. 출력 헤더와 함께 표 형식으 로 보여줌.)-a : free 프레임까지 포함해서 모두 출력. -p <PID> : 지정 PID의 프레임만 출력.  **핵심 동작 흐름  
 1.** 옵션 파싱(없거나 이상하면 usage).  **2.** dump\_physmem\_info(buf, MAX\_FRINFO)로 **커널 PF 테이블 스냅샷** 획득. 실패 시 에러.  **3.** 루프에서 필터링 적용: -a 없으면 alloc==1만, -p 있으면 pid 일치만 출력.  **4.** 한 프레임당 한 줄로 frame#, alloc, pid, start\_tick 출력.

**커널/시스템콜 연계 포인트 :** 직접 호출하는 시스템콜은 **dump\_physmem\_info()** 하나이며, 이 콜은 커널의 pf\_info[](물리 프레임 회계)에서 allocated/pid/start\_tick 스냅샷을 만들어 유저 버퍼로 copyout합니다.

memtest.c

텍스트, 스크린샷, 메뉴, 바이올렛색이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

여러 프로세스를 순차적으로 생성하여 \*\*메모리 할당 부하(memstress)\*\*를 만들고, 중간중간 memdump로 특정 PID의 프레임 상태를 관찰하는 통합 시나리오 테스트입니다.

1. 자식 #1: memstress -n 31 -t 500 실행(약 500 ticks 유지)
2. sleep(100) - #1이 어느 정도 메모리를 잡고 있는 상태 보장
3. 자식 #2: 동일한 memstress 또 실행(동시/시차 점유 유도)
4. sleep(100) - 두 memstress가 모두 살아있는 시점 확보
5. 자식 #3: memdump -p 4 실행(#1의 PID가 4라는 가정; 실제 PID는 부팅 상황마다 달라질 수 있으므로 참고)
6. sleep(100) 후, 자식 #4: memdump -p 5 실행(#2의 PID 가정)
7. wait() 4회로 앞선 자식 종료 수거(정상화)
8. sleep(100) 후, 자식 #5: 다시 memdump -p 5 (종료 이후에 프레임이 정리되었는지 확인)
9. 마지막 wait() 후 종료.

makefile

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

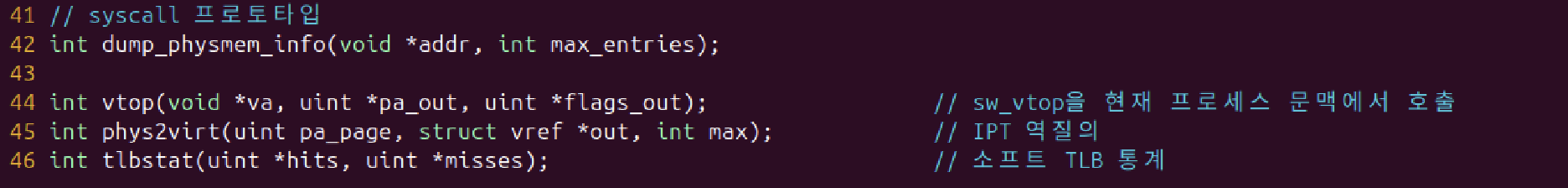
스크린샷, 텍스트, 라일락이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

스크린샷, 텍스트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

C 추가구현 부분 -> 아래에 자세히 명시되어있지만(테스트: make qemu -> projtest = vtop, pfind 사용 테스트프로그램)  
(user.h)

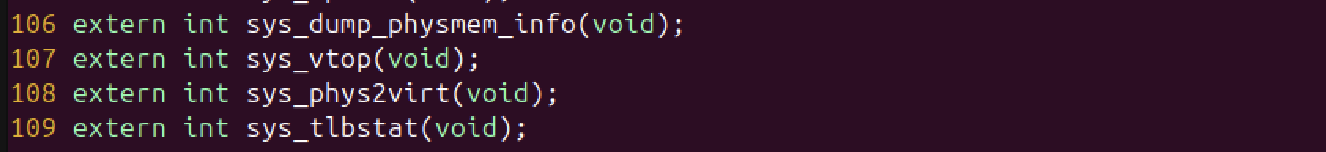
(types.h)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.  
(syscall.h)

텍스트, 폰트, 스크린샷, 바이올렛색이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.  
(syscall.c)

  
텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.  
(usys.S)

  
(defs.h)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

fork()를 COW(copy-on-write) 로 바꿔서, 동일 물리페이지를 부모/자식이 공유하도록 만들고, 쓰기 시점에만 복사되게 하면 pfind로 IPT 중복 체인확인

(mmu.h) : 임의로 써도 되는 **AVAIL 비트**중 하나를 **COW 표시용으로 사용**

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(vm.c) cow\_share\_page() : fork()에서 복사 대신 공유로 바꾸기위한 매핑함수

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(vm.c) cowuvm() : 전체 주소공간을 공유로 복제

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(proc.c) fork() : copyuvm() → cowuvm()으로 교체

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

ㄴ xv6의 fork()를 진짜 COW(copy-on-write) 로 바꿔서, 동일 물리페이지를 부모/자식이 공유하도록 만들고, 쓰기 시점에만 복사되게 하면 pfind로 IPT 중복 체인을 보일 수 있다

(proc.c) wait() - 자식 pid의 STLB/ IPT를 먼저 비우고 → freevm(child->pgdir)을 호출

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(vm.c) cow\_fault() : COW 폴트 처리 함수

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(trap.c) 트랩에서 page fault잡기

텍스트, 스크린샷, 폰트, 바이올렛색이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(defs.h) COW핸들러 프로토타입 추가



(vm.c) sw\_vtop + soft TLB + IPT 구현

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(vm.c) soft TLB invalidate/update API 추가

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(IPT flages 갱신 API 추가)

텍스트, 스크린샷이(가) 표시된 사진

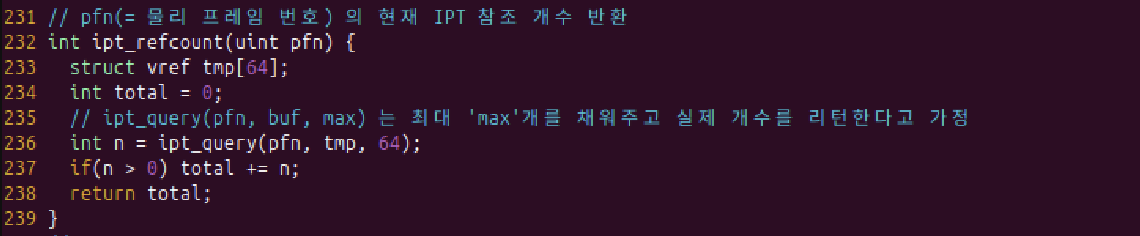
AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(exit 시점에 즉시 IPT 지우기)

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

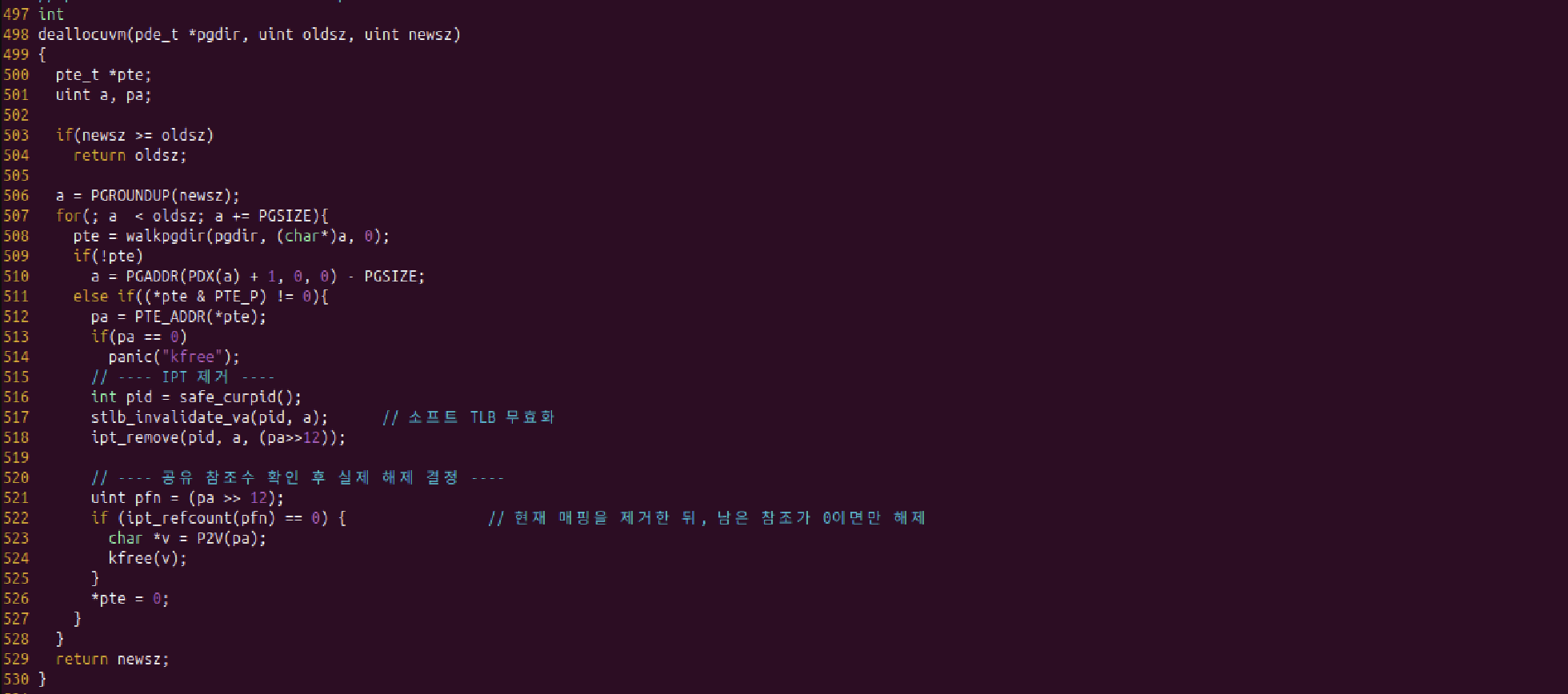
(참조수 헬퍼 추가)



텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(kfree 전에 참조수 확인)



(vm.c - clearpteu) 권한 변경 지점에서 IPT flags 갱신 API(ipt\_update\_flags)호출

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(sysproc.c) 새 시스템콜 3종

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

exec/exit 시 현재 PID의 소프트 TLB 엔트리 비우기

(proc.c - exit()) - exit() 이후엔 같은 PID가 더 이상 실행되지 않아야 됨, 이 타이밍에 purge해 다른 코드와 충돌 없이 정리

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(exec.c - exec()) - 같은 PID가 완전히 새 pgdir을 쓰기 시작했으니, 이전 주소공간에 대한 TLB 엔트리를 버려야 함

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(main.c) 초기화 연동 - 부팅 완료 시점에 호출(락 준비된 뒤)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

사용자 유틸 2종(vtop, pfind)

(vtop.c)

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.vtop.c : 현재 프로세스의 VA→PA 확인 + SW-TLB 히트/미스 관찰 (기본형태 : $ vtop <hex\_va>)

커널에서 sys\_vtop()을 호출해 소프트웨어 페이지 워커(+ STLB)를 거친 결과를 출력.(PA, flags, TLB[..]stlb누적 통계(hit/miss))

편의옵션:  
 -s : 더미변수 주소(스택)으로 빠르게 테스트/디버깅 가능  
 -r <N> : 같은 주소 N번 반복 조회(첫 번째는 miss, 두 번째부터 hit으로 올라가는 것 확인 / 동일 프로세스, 동일 VA)  
 -a <pages> : 힙을 N페이지 확장후 바로 조회(sbrk(pages\*PGSIZE)로 N 페이지를 확보, 각 페이지 첫 바이트를 1회 접근, 첫 페이지의 VA(base)를 대상으로 vtop()을 반복 호출해 STLB 채워지는 과정을 한 번에 확인.

+ 미매핑 케이스 -> vtop: not present or invalid VA

(pfind.c) - 물리 페이지에 매핑된 (pid,va,flags)들을 나열

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.pfind.c : 물리 페이지(페이지 베이스)로 역질의(IPT) (기본형태 : $ pfind 0x<hex\_pa\_page>)

인자로 넘긴 주소는 프로그램 내부에서 & ~0xFFF로 페이지 정렬되므로, 페이지 안의 어느 오프셋을 줘도 같은 페이지로 처리된다.  
 - N refs는 해당 PFN으로 매핑된 (pid, va) 엔트리 수.  
 - IPT는 유저 매핑(PTE\_U=1)만 추적하므로 커널 전용 매핑은 나오지 않는다.  
 - 한 페이지가 여러 프로세스에서 공유되면(CoW나 중복 매핑) 레코드가 여러 줄로 나온다.

0 refs가 나오는 경우 :  
 **-** 해당 PFN을 아무 유저 프로세스도 매핑하지 않을 때(해제됨, 혹은 커널 전용 매핑만 있을 때). **-** 프로세스가 exit() 되어 ipt\_purge\_pid()로 정리된 직후.

(makefile)

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

(projtest.c) C를 위한 테스트 프로그램

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.  
텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

projtest 실행

1. 소프트 TLB(STLB) 캐시가 제대로 동작하는지,
2. sbrk로 만든 페이지를 해제할 때 IPT/STLB가 일관되게 정리되는지,
3. COW(copy-on-write)에서 PFN 공유→분리→종료 후 정리가 정확한지  
   를 **vtop/pfind 유틸을 실제로 fork+exec** 해서 눈으로 확인하게 해주는 통합 테스트 프로그램.

**구성 요소**

**1) 헬퍼 함수  
 -** to\_hex(uint x, char \*buf) 32비트 값을 0x........ 형태의 문자열로 바꿔서 로그에 예쁘게 찍기 위한 유틸. **-** run(const char \*prog, char \*const argv[]) fork() 후 자식에서 exec(prog, argv)를 호출하고, 부모는 wait()로 끝까지 기다리는 간단한 실행 래퍼. → vtop, pfind를 **진짜 사용자 프로그램**으로 실행해서 그들의 출력(예: PFN/refs/flags, STLB hit/miss)을 그대로 보게 함.

**2) 메인 시나리오(main)**

헤더를 출력한 다음 A→B→C 순서로 검증한다.

**[A] STLB hit/miss 확인  
 -** run("vtop", {"vtop","-s","-r","3",0}) 를 실행.  
vtop -s -r 3는 같은 VA를 3번 연속 조회하며 STLB 통계를 함께 출력한다. **-** 기대 출력:  
1회차 miss, 2·3회차 hit → TLB[hits=0→1→2, misses=1].

**[B] remap/unmap → IPT/STLB 일관성**

1. char \*base = sbrk(PGSIZE\*2);로 두 페이지 확보 후 각 페이지를 1바이트씩 터치(유효 매핑 생성).
2. **syscall vtop(base,&pa,&flags)** 로 base의 물리주소를 구하고, PA\_PAGE(페이지 단위 PFN 주소)로 정규화.
3. run("vtop", {"vtop", VA\_hex, 0})로 해당 VA가 present임을 확인,  
   run("pfind", {"pfind", PFN\_hex, 0})로 \*\*현재 PFN 참조수가 1(부모만)\*\*임을 확인.
4. sbrk(-PGSIZE\*2)로 두 페이지를 해제.
5. 다시 pfind PFN → **0 refs**, vtop VA → **not present**가 떠야 한다.  
   (여기서 **PTE=0 → HW TLB flush → STLB/ IPT 정리** 순서가 맞지 않으면 어긋난다.)

**[C] COW 공유 → 쓰기 분리 → exit 정리**

1. 부모가 sbrk(PGSIZE)로 1페이지 확보, vtop(syscall)로 원래 PFN을 구해둠.
2. fork():  
   **자식**: 잠깐 sleep()으로 대기 → p[0]=99;로 쓰기 → COW 발생(새 PFN 배정) → 로그 출력 → exit().  
   **부모**:  
    - 자식 쓰기 전 pfind(원래 PFN) → **2 refs**(부모+자식).  
    - 자식 쓰기 후(부모는 sleep()으로 타이밍 맞춤) 다시 pfind(원래 PFN) → **1 refs**(자식이 분리).  
    - wait()로 자식 종료 대기 후 다시 pfind(원래 PFN) → **여전히 1 refs**(부모만 남음).  
    - 마지막으로 부모가 sbrk(-PGSIZE)로 해제 → pfind(원래 PFN) → **0 refs**.

포인트: COW 이후에는 “원래 PFN”은 부모만 유지하므로 1 refs가 되고, 자식이 받은 “새 PFN”은 자식 exit() 후 0 refs가 되는 게 정상. projtest는 원래 PFN만 추적해도 일관성을 확인할 수 있게 구성

C-테스트 (projtest 실행)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**4. 주석 달린 소스코드 :**

<memstress.c>

#include "types.h"

#include "stat.h"

#include "user.h"

#include "fcntl.h"

static void

usage(void) {

printf(1, "usage: memstress [-n pages] [-t ticks] [-w]\n");

exit();

}

int

main(int *argc*, char \**argv*[])

{

int pages = 10; *// 기본: 10 페이지*

int hold\_ticks = 200; *// 기본: 200 ticks 유지*

int do\_write = 0; *// 기본: 쓰기 미수행*

*// 옵션 파싱*

for (int i = 1; i < argc; i++) {

if (!strcmp(argv[i], "-n")) {

if (i + 1 >= argc) usage();

pages = atoi(argv[++i]);

if (pages <= 0) usage();

} else if (!strcmp(argv[i], "-t")) {

if (i + 1 >= argc) usage();

hold\_ticks = atoi(argv[++i]);

if (hold\_ticks < 0) usage();

} else if (!strcmp(argv[i], "-w")) {

do\_write = 1;

} else {

usage();

}

}

int pid = getpid();

printf(1, "[memstress] pid=%d pages=%d hold=%d ticks write=%d\n",

pid, pages, hold\_ticks, do\_write);

*// 메모리 확보*

int inc = pages \* 4096;

char \*base = sbrk(inc);

if (base == (char\*)-1) {

printf(1, "[memstress] sbrk failed\n");

exit();

}

*// 실제 접근(페이지 당 1바이트)으로 물리 할당 유도*

if (do\_write) {

for (int p = 0; p < pages; p++) {

base[p \* 4096] = (char)(p & 0xff);

}

}

*// 유지 시간*

sleep(hold\_ticks);

printf(1, "[memstress] pid=%d done\n", pid);

exit();

}

<memdump.c>

#include "types.h"

#include "stat.h"

#include "user.h"

#include "fcntl.h"

#define MAX\_FRINFO 60000

static void

usage(void)

{

printf(1, "usage: memdump [-a] [-p PID]\n");

exit();

}

int

main(int *argc*, char \**argv*[])

{

int show\_all = 0; *// -a가 있을 때만 free 프레임까지 표시*

int filter\_pid = -1; *// -p로 지정된 PID만 표시(기본: 필터 없음)*

if (argc == 1) usage();

*// 옵션 파싱*

for (int i = 1; i < argc; i++) {

if (!strcmp(argv[i], "-a")) {

show\_all = 1;

} else if (!strcmp(argv[i], "-p")) {

if (i + 1 >= argc) usage();

filter\_pid = atoi(argv[++i]);

} else {

usage();

}

}

static struct physframe\_info buf[MAX\_FRINFO];

int n = dump\_physmem\_info((void \*)buf, MAX\_FRINFO);

if (n < 0) {

printf(1, "memdump: dump\_physmem\_info failed\n");

exit();

}

printf(1, "[memdump] pid=%d\n", getpid());

printf(1, "[frame#]\t[alloc]\t[pid]\t[start\_tick]\n");

for (int i = 0; i < n; i++) {

int alloc = buf[i].allocated;

int pid = buf[i].pid;

*// 기본은 "할당된 것만" 출력, -a면 free까지 포함*

if (!show\_all && alloc == 0) continue;

*// -p가 있으면 해당 PID만 출력(보통 alloc==1인 것만 의미 있음)*

if (filter\_pid != -1 && pid != filter\_pid) continue;

printf(1, "%d\t\t%d\t%d\t%d\n",

buf[i].frame\_index, alloc, pid, buf[i].start\_tick);

}

exit();

}

<memtest.c>

#include "types.h"

#include "stat.h"

#include "user.h"

int

main(int *argc*, char \**argv*[])

{

int pid;

pid = fork();

if(pid < 0){

printf(1, "fork failed\n");

exit();

}

if(pid == 0){

char \*args[] = { "memstress", "-n", "31", "-t", "500", 0 };

exec("memstress", args);

printf(1, "exec memstress failed\n");

exit();

}

sleep(100);

int pid2 = fork();

if(pid2 == 0){

char \*args2[] = { "memstress", "-n", "31", "-t", "500", 0 };

exec("memstress", args2);

printf(1, "exec memstress failed\n");

exit();

}

sleep(100);

int pid3 = fork();

if(pid3 < 0){

printf(1, "fork failed\n");

exit();

}

if(pid3 == 0){

char \*args3[] = { "memdump", "-p", "4", 0 };

exec("memdump", args3);

printf(1, "exec memdump failed\n");

exit();

}

sleep(100);

int pid4 = fork();

if(pid4 < 0){

printf(1, "fork failed\n");

exit();

}

if(pid4 == 0){

char \*args4[] = { "memdump", "-p", "5", 0 };

exec("memdump", args4);

printf(1, "exec memdump failed\n");

exit();

}

wait();

wait();

wait();

wait();

sleep(100);

int pid5 = fork();

if(pid5 < 0){

printf(1, "fork failed\n");

exit();

}

if(pid5 == 0){

char \*args5[] = { "memdump", "-p", "5", 0 };

exec("memdump", args5);

printf(1, "exec memdump failed\n");

exit();

}

wait();

exit();

}

<types.h>

typedef unsigned int uint;

typedef unsigned short ushort;

typedef unsigned char uchar;

typedef uint pde\_t;

struct physframe\_info {

uint frame\_index; *// PFN*

int allocated; *// 1: in use, 0: free*

int pid; *// owner pid, kernel/none: -1*

uint start\_tick; *// first-use tick*

};

struct vref {

uint pid; *// 소유 PID*

uint va; *// 페이지 단위 가상주소 (PGROUNDDOWN)*

uint flags; *// PTE 권한 스냅샷*

};

<user.h>

struct stat;

struct rtcdate;

*// system calls*

int fork(void);

int exit(void) \_\_attribute\_\_((noreturn));

int wait(void);

int pipe(int\*);

int write(int, const void\*, int);

int read(int, void\*, int);

int close(int);

int kill(int);

int exec(char\*, char\*\*);

int open(const char\*, int);

int mknod(const char\*, short, short);

int unlink(const char\*);

int fstat(int *fd*, struct stat\*);

int link(const char\*, const char\*);

int mkdir(const char\*);

int chdir(const char\*);

int dup(int);

int getpid(void);

char\* sbrk(int);

int sleep(int);

int uptime(void);

*// ulib.c*

int stat(const char\*, struct stat\*);

char\* strcpy(char\*, const char\*);

void \*memmove(void\*, const void\*, int);

char\* strchr(const char\*, char *c*);

int strcmp(const char\*, const char\*);

void printf(int, const char\*, ...);

char\* gets(char\*, int *max*);

uint strlen(const char\*);

void\* memset(void\*, int, uint);

void\* malloc(uint);

void free(void\*);

int atoi(const char\*);

*// syscall 프로토타입*

int dump\_physmem\_info(void \**addr*, int *max\_entries*);

int vtop(void \**va*, uint \**pa\_out*, uint \**flags\_out*); *// sw\_vtop을 현재 프로세스 문맥에서 호출*

int phys2virt(uint *pa\_page*, struct vref \**out*, int *max*); *// IPT 역질의*

int tlbstat(uint \**hits*, uint \**misses*); *// 소프트 TLB 통계*

<usys.S>

#include "syscall.h"

#include "traps.h"

#define SYSCALL(name) \

.globl name; \

name: \

movl $SYS\_ ## name, %eax; \

int $T\_SYSCALL; \

ret

SYSCALL(fork)

SYSCALL(exit)

SYSCALL(wait)

SYSCALL(pipe)

SYSCALL(read)

SYSCALL(write)

SYSCALL(close)

SYSCALL(kill)

SYSCALL(exec)

SYSCALL(open)

SYSCALL(mknod)

SYSCALL(unlink)

SYSCALL(fstat)

SYSCALL(link)

SYSCALL(mkdir)

SYSCALL(chdir)

SYSCALL(dup)

SYSCALL(getpid)

SYSCALL(sbrk)

SYSCALL(sleep)

SYSCALL(uptime)

SYSCALL(dump\_physmem\_info)

SYSCALL(vtop)

SYSCALL(phys2virt)

SYSCALL(tlbstat)

<syscall.h>

*// System call numbers*

#define SYS\_fork 1

#define SYS\_exit 2

#define SYS\_wait 3

#define SYS\_pipe 4

#define SYS\_read 5

#define SYS\_kill 6

#define SYS\_exec 7

#define SYS\_fstat 8

#define SYS\_chdir 9

#define SYS\_dup 10

#define SYS\_getpid 11

#define SYS\_sbrk 12

#define SYS\_sleep 13

#define SYS\_uptime 14

#define SYS\_open 15

#define SYS\_write 16

#define SYS\_mknod 17

#define SYS\_unlink 18

#define SYS\_link 19

#define SYS\_mkdir 20

#define SYS\_close 21

#define SYS\_dump\_physmem\_info 22

#define SYS\_vtop 23

#define SYS\_phys2virt 24

#define SYS\_tlbstat 25

<syscall.c>

#include "types.h"

#include "defs.h"

#include "param.h"

#include "memlayout.h"

#include "mmu.h"

#include "proc.h"

#include "x86.h"

#include "syscall.h"

*// User code makes a system call with INT T\_SYSCALL.*

*// System call number in %eax.*

*// Arguments on the stack, from the user call to the C*

*// library system call function. The saved user %esp points*

*// to a saved program counter, and then the first argument.*

*// Fetch the int at addr from the current process.*

int

fetchint(uint *addr*, int \**ip*)

{

struct proc \*curproc = myproc();

if(addr >= curproc->sz || addr+4 > curproc->sz)

return -1;

\*ip = \*(int\*)(addr);

return 0;

}

*// Fetch the nul-terminated string at addr from the current process.*

*// Doesn't actually copy the string - just sets \*pp to point at it.*

*// Returns length of string, not including nul.*

int

fetchstr(uint *addr*, char \*\**pp*)

{

char \*s, \*ep;

struct proc \*curproc = myproc();

if(addr >= curproc->sz)

return -1;

\*pp = (char\*)addr;

ep = (char\*)curproc->sz;

for(s = \*pp; s < ep; s++){

if(\*s == 0)

return s - \*pp;

}

return -1;

}

*// Fetch the nth 32-bit system call argument.*

int

argint(int *n*, int \**ip*)

{

return fetchint((myproc()->tf->esp) + 4 + 4\*n, ip);

}

*// Fetch the nth word-sized system call argument as a pointer*

*// to a block of memory of size bytes. Check that the pointer*

*// lies within the process address space.*

int

argptr(int *n*, char \*\**pp*, int *size*)

{

int i;

struct proc \*curproc = myproc();

if(argint(n, &i) < 0)

return -1;

if(size < 0 || (uint)i >= curproc->sz || (uint)i+size > curproc->sz)

return -1;

\*pp = (char\*)i;

return 0;

}

*// Fetch the nth word-sized system call argument as a string pointer.*

*// Check that the pointer is valid and the string is nul-terminated.*

*// (There is no shared writable memory, so the string can't change*

*// between this check and being used by the kernel.)*

int

argstr(int *n*, char \*\**pp*)

{

int addr;

if(argint(n, &addr) < 0)

return -1;

return fetchstr(addr, pp);

}

extern int sys\_chdir(void);

extern int sys\_close(void);

extern int sys\_dup(void);

extern int sys\_exec(void);

extern int sys\_exit(void);

extern int sys\_fork(void);

extern int sys\_fstat(void);

extern int sys\_getpid(void);

extern int sys\_kill(void);

extern int sys\_link(void);

extern int sys\_mkdir(void);

extern int sys\_mknod(void);

extern int sys\_open(void);

extern int sys\_pipe(void);

extern int sys\_read(void);

extern int sys\_sbrk(void);

extern int sys\_sleep(void);

extern int sys\_unlink(void);

extern int sys\_wait(void);

extern int sys\_write(void);

extern int sys\_uptime(void);

extern int sys\_dump\_physmem\_info(void);

extern int sys\_vtop(void);

extern int sys\_phys2virt(void);

extern int sys\_tlbstat(void);

static int (\*syscalls[])(void) = {

[SYS\_fork] sys\_fork,

[SYS\_exit] sys\_exit,

[SYS\_wait] sys\_wait,

[SYS\_pipe] sys\_pipe,

[SYS\_read] sys\_read,

[SYS\_kill] sys\_kill,

[SYS\_exec] sys\_exec,

[SYS\_fstat] sys\_fstat,

[SYS\_chdir] sys\_chdir,

[SYS\_dup] sys\_dup,

[SYS\_getpid] sys\_getpid,

[SYS\_sbrk] sys\_sbrk,

[SYS\_sleep] sys\_sleep,

[SYS\_uptime] sys\_uptime,

[SYS\_open] sys\_open,

[SYS\_write] sys\_write,

[SYS\_mknod] sys\_mknod,

[SYS\_unlink] sys\_unlink,

[SYS\_link] sys\_link,

[SYS\_mkdir] sys\_mkdir,

[SYS\_close] sys\_close,

[SYS\_dump\_physmem\_info] sys\_dump\_physmem\_info,

[SYS\_vtop] sys\_vtop,

[SYS\_phys2virt] sys\_phys2virt,

[SYS\_tlbstat] sys\_tlbstat,

};

void

syscall(void)

{

int num;

struct proc \*curproc = myproc();

num = curproc->tf->eax;

if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {

curproc->tf->eax = syscalls[num]();

} else {

cprintf("%d %s: unknown sys call %d\n",

curproc->pid, curproc->name, num);

curproc->tf->eax = -1;

}

}

<sysproc.c>

#include "types.h"

#include "x86.h"

#include "defs.h"

#include "date.h"

#include "param.h"

#include "memlayout.h"

#include "mmu.h"

#include "proc.h"

#include "spinlock.h"

extern struct {

struct spinlock lock;

int use\_lock;

struct run \*freelist;

} kmem;

int sys\_dump\_physmem\_info(void);

int

sys\_fork(void)

{

return fork();

}

int

sys\_exit(void)

{

exit();

return 0; *// not reached*

}

int

sys\_wait(void)

{

return wait();

}

int

sys\_kill(void)

{

int pid;

if(argint(0, &pid) < 0)

return -1;

return kill(pid);

}

int

sys\_getpid(void)

{

return myproc()->pid;

}

int

sys\_sbrk(void)

{

int addr;

int n;

if(argint(0, &n) < 0)

return -1;

addr = myproc()->sz;

if(growproc(n) < 0)

return -1;

return addr;

}

int

sys\_sleep(void)

{

int n;

uint ticks0;

if(argint(0, &n) < 0)

return -1;

acquire(&tickslock);

ticks0 = ticks;

while(ticks - ticks0 < n){

if(myproc()->killed){

release(&tickslock);

return -1;

}

sleep(&ticks, &tickslock);

}

release(&tickslock);

return 0;

}

*// return how many clock tick interrupts have occurred*

*// since start.*

int

sys\_uptime(void)

{

uint xticks;

acquire(&tickslock);

xticks = ticks;

release(&tickslock);

return xticks;

}

extern struct physframe\_info pf\_info[]; *// 전역 테이블*

extern const int PFN\_TABLE\_SIZE;

int

sys\_dump\_physmem\_info(void)

{

char \*uaddr;

int maxe;

if(argptr(0, &uaddr, sizeof(struct physframe\_info\*)) < 0) return -1;

if(argint(1, &maxe) < 0) return -1;

if(maxe <= 0) return 0;

if (maxe > PFN\_TABLE\_SIZE) maxe = PFN\_TABLE\_SIZE;

*// 스냅샷 일관성 보장: kmem.lock으로 보호*

acquire(&kmem.lock);

int n = maxe;

*// 연속 복사*

int ok = copyout(myproc()->pgdir, (uint)uaddr,

(void\*)pf\_info, sizeof(struct physframe\_info)\*n);

release(&kmem.lock);

if(ok < 0) return -1;

return n;

}

*// vtop: 현재 프로세스 pgdir 기준*

int sys\_vtop(void){

char \*u\_va;

char \*u\_pa\_out, \*u\_flags\_out;

if(argptr(0, &u\_va, 1) < 0) return -1;

if(argptr(1, &u\_pa\_out, sizeof(uint)) < 0) return -1;

if(argptr(2, &u\_flags\_out, sizeof(uint)) < 0) return -1;

uint va = (uint)u\_va;

uint pa, flags;

int pid = myproc() ? myproc()->pid : -1;

uint va\_page = PGROUNDDOWN(va);

*// Soft TLB 조회 → 미스면 sw\_vtop 수행 후 채움*

uint pa\_page;

if(!stlb\_lookup(pid, va\_page, &pa\_page, &flags)){

if(sw\_vtop(myproc()->pgdir, (void\*)va, &pa, &flags) < 0) return -1;

pa\_page = pa & ~0xFFF;

stlb\_fill(pid, va\_page, pa\_page, flags);

}else{

pa = pa\_page | (va & 0xFFF);

}

if(copyout(myproc()->pgdir, (uint)u\_pa\_out, (char\*)&pa, sizeof(pa)) < 0) return -1;

if(copyout(myproc()->pgdir, (uint)u\_flags\_out, (char\*)&flags, sizeof(flags)) < 0) return -1;

return 0;

}

*// phys2virt: pfn 역질의*

int sys\_phys2virt(void){

int pa\_page, max;

char \*u\_out;

if(argint(0, &pa\_page) < 0) return -1;

if(argint(2, &max) < 0) return -1; *// ← 먼저 max를 읽는다*

if(max <= 0) return 0;

if(max > 64) max = 64;

*// 이제 max를 반영해 사용자 버퍼 검증*

if(argptr(1, &u\_out, sizeof(struct vref) \* max) < 0) return -1;

struct vref kbuf[64];

int n = ipt\_query((uint)pa\_page >> 12, kbuf, max);

if(n <= 0) return 0;

*// --- 유저 페이지(PTE\_U)만 남기기: in-place compact ---*

int m = 0;

for(int i = 0; i < n; i++){

if(kbuf[i].flags & PTE\_U) *// 유저 비트 있는 것만*

kbuf[m++] = kbuf[i]; *// 앞으로 땡겨서 보관*

}

*// m <= max 이므로 사용자 버퍼 검증은 기존 max 기준이면 충분*

int bytes = sizeof(struct vref) \* m;

if(copyout(myproc()->pgdir, (uint)u\_out, (char\*)kbuf, bytes) < 0) return -1;

return m; *// 필터링 후 개수 반환*

}

*// tlbstat: 히트/미스 통계*

int sys\_tlbstat(void){

char \*u\_hits, \*u\_misses;

if(argptr(0, &u\_hits, sizeof(uint)) < 0) return -1;

if(argptr(1, &u\_misses, sizeof(uint)) < 0) return -1;

uint h, m; stlb\_stats(&h, &m);

if(copyout(myproc()->pgdir, (uint)u\_hits, (char\*)&h, sizeof(h)) < 0) return -1;

if(copyout(myproc()->pgdir, (uint)u\_misses, (char\*)&m, sizeof(m)) < 0) return -1;

return 0;

}

<kalloc.c>

*// Physical memory allocator, intended to allocate*

*// memory for user processes, kernel stacks, page table pages,*

*// and pipe buffers. Allocates 4096-byte pages.*

#include "types.h"

#include "defs.h"

#include "param.h"

#include "memlayout.h"

#include "mmu.h"

#include "spinlock.h"

#include "proc.h"

#include "date.h"

extern int pf\_ready;

*// ---- 전역 프레임 추적 테이블 ----*

#define PFNNUM 60000 *// 테이블 크기*

const int PFN\_TABLE\_SIZE = PFNNUM; *// 다른 파일에서 extern으로 참조할 변수 이름*

struct physframe\_info pf\_info[PFNNUM];

static inline uint pa\_to\_pfn(uint *pa*){ return pa >> 12; }

static inline void pf\_reset(uint *pfn*){

if(pfn >= PFNNUM) return;

pf\_info[pfn].frame\_index = pfn;

pf\_info[pfn].allocated = 0;

pf\_info[pfn].pid = -1;

pf\_info[pfn].start\_tick = 0;

}

*// 초기화 루틴(한 번만 호출)*

static void pfinfo\_init\_once(void){

for(int i=0;i<PFNNUM;i++){

pf\_info[i].frame\_index = i;

pf\_info[i].allocated = 0;

pf\_info[i].pid = -1;

pf\_info[i].start\_tick = 0;

}

}

void freerange(void \**vstart*, void \**vend*);

extern char end[]; *// first address after kernel loaded from ELF file*

*// defined by the kernel linker script in kernel.ld*

struct run {

struct run \*next;

};

struct {

struct spinlock lock;

int use\_lock;

struct run \*freelist;

} kmem;

*// Initialization happens in two phases.*

*// 1. main() calls kinit1() while still using entrypgdir to place just*

*// the pages mapped by entrypgdir on free list.*

*// 2. main() calls kinit2() with the rest of the physical pages*

*// after installing a full page table that maps them on all cores.*

void

kinit1(void \**vstart*, void \**vend*)

{

initlock(&kmem.lock, "kmem");

kmem.use\_lock = 0;

pfinfo\_init\_once(); *// pf\_info[] 전체 초기화는 여기서 한 번만*

freerange(vstart, vend);

}

void

kinit2(void \**vstart*, void \**vend*)

{

freerange(vstart, vend);

kmem.use\_lock = 1;

}

void

freerange(void \**vstart*, void \**vend*)

{

char \*p;

p = (char\*)PGROUNDUP((uint)vstart);

for(; p + PGSIZE <= (char\*)vend; p += PGSIZE)

kfree(p);

}

*//PAGEBREAK: 21*

*// Free the page of physical memory pointed at by v,*

*// which normally should have been returned by a*

*// call to kalloc(). (The exception is when*

*// initializing the allocator; see kinit above.)*

void

kfree(char \**v*)

{

struct run \*r;

if((uint)v % PGSIZE || v < end || V2P(v) >= PHYSTOP)

panic("kfree");

*// poison & freelist 삽입*

memset(v, 1, PGSIZE);

r = (struct run\*)v;

if(kmem.use\_lock) acquire(&kmem.lock);

r->next = kmem.freelist;

kmem.freelist = r;

if(kmem.use\_lock) release(&kmem.lock);

}

*// Allocate one 4096-byte page of physical memory.*

*// Returns a pointer that the kernel can use.*

*// Returns 0 if the memory cannot be allocated.*

extern uint ticks;

extern struct spinlock tickslock;

char\*

kalloc(void)

{

struct run \*r;

if(kmem.use\_lock) acquire(&kmem.lock);

r = kmem.freelist;

if(r) kmem.freelist = r->next;

if(kmem.use\_lock) release(&kmem.lock);

if(r){

uint pa = V2P((char\*)r);

uint pfn = pa\_to\_pfn(pa);

if(!kmem.use\_lock){

*// ----- 부팅 초기: CPU/틱 미초기화, 안전하게 최소 기록만 -----*

pf\_info[pfn].allocated = 1;

pf\_info[pfn].pid = -1; *// 아직 소유 프로세스 개념 없음*

pf\_info[pfn].start\_tick = 0;

pf\_info[pfn].frame\_index= pfn;

} else {

*// ----- 정상 운행 이후 -----*

acquire(&kmem.lock);

pf\_info[pfn].allocated = 1;

struct proc \*p = myproc(); *// 이제 안전*

pf\_info[pfn].pid = p ? p->pid : -1;

acquire(&tickslock);

pf\_info[pfn].start\_tick = ticks;

release(&tickslock);

pf\_info[pfn].frame\_index = pfn;

release(&kmem.lock);

}

}

return (char\*)r;

}

<defs.h>

struct buf;

struct context;

struct file;

struct inode;

struct pipe;

struct proc;

struct rtcdate;

struct spinlock;

struct sleeplock;

struct stat;

struct superblock;

*// bio.c*

void binit(void);

struct buf\* bread(uint, uint);

void brelse(struct buf\*);

void bwrite(struct buf\*);

*// console.c*

void consoleinit(void);

void cprintf(char\*, ...);

void consoleintr(int(\*)(void));

void panic(char\*) \_\_attribute\_\_((noreturn));

*// exec.c*

int exec(char\*, char\*\*);

*// file.c*

struct file\* filealloc(void);

void fileclose(struct file\*);

struct file\* filedup(struct file\*);

void fileinit(void);

int fileread(struct file\*, char\*, int *n*);

int filestat(struct file\*, struct stat\*);

int filewrite(struct file\*, char\*, int *n*);

*// fs.c*

void readsb(int *dev*, struct superblock \**sb*);

int dirlink(struct inode\*, char\*, uint);

struct inode\* dirlookup(struct inode\*, char\*, uint\*);

struct inode\* ialloc(uint, short);

struct inode\* idup(struct inode\*);

void iinit(int *dev*);

void ilock(struct inode\*);

void iput(struct inode\*);

void iunlock(struct inode\*);

void iunlockput(struct inode\*);

void iupdate(struct inode\*);

int namecmp(const char\*, const char\*);

struct inode\* namei(char\*);

struct inode\* nameiparent(char\*, char\*);

int readi(struct inode\*, char\*, uint, uint);

void stati(struct inode\*, struct stat\*);

int writei(struct inode\*, char\*, uint, uint);

*// ide.c*

void ideinit(void);

void ideintr(void);

void iderw(struct buf\*);

*// ioapic.c*

void ioapicenable(int *irq*, int *cpu*);

extern uchar ioapicid;

void ioapicinit(void);

*// kalloc.c*

char\* kalloc(void);

void kfree(char\*);

void kinit1(void\*, void\*);

void kinit2(void\*, void\*);

*// kbd.c*

void kbdintr(void);

*// lapic.c*

void cmostime(struct rtcdate \**r*);

int lapicid(void);

extern volatile uint\* lapic;

void lapiceoi(void);

void lapicinit(void);

void lapicstartap(uchar, uint);

void microdelay(int);

*// log.c*

void initlog(int *dev*);

void log\_write(struct buf\*);

void begin\_op();

void end\_op();

*// mp.c*

extern int ismp;

void mpinit(void);

*// picirq.c*

void picenable(int);

void picinit(void);

*// pipe.c*

int pipealloc(struct file\*\*, struct file\*\*);

void pipeclose(struct pipe\*, int);

int piperead(struct pipe\*, char\*, int);

int pipewrite(struct pipe\*, char\*, int);

*//PAGEBREAK: 16*

*// proc.c*

int cpuid(void);

void exit(void);

int fork(void);

int growproc(int);

int kill(int);

struct cpu\* mycpu(void);

struct proc\* myproc();

void pinit(void);

void procdump(void);

void scheduler(void) \_\_attribute\_\_((noreturn));

void sched(void);

void setproc(struct proc\*);

void sleep(void\*, struct spinlock\*);

void userinit(void);

int wait(void);

void wakeup(void\*);

void yield(void);

*// swtch.S*

void swtch(struct context\*\*, struct context\*);

*// spinlock.c*

void acquire(struct spinlock\*);

void getcallerpcs(void\*, uint\*);

int holding(struct spinlock\*);

void initlock(struct spinlock\*, char\*);

void release(struct spinlock\*);

void pushcli(void);

void popcli(void);

*// sleeplock.c*

void acquiresleep(struct sleeplock\*);

void releasesleep(struct sleeplock\*);

int holdingsleep(struct sleeplock\*);

void initsleeplock(struct sleeplock\*, char\*);

*// string.c*

int memcmp(const void\*, const void\*, uint);

void\* memmove(void\*, const void\*, uint);

void\* memset(void\*, int, uint);

char\* safestrcpy(char\*, const char\*, int);

int strlen(const char\*);

int strncmp(const char\*, const char\*, uint);

char\* strncpy(char\*, const char\*, int);

*// syscall.c*

int argint(int, int\*);

int argptr(int, char\*\*, int);

int argstr(int, char\*\*);

int fetchint(uint, int\*);

int fetchstr(uint, char\*\*);

void syscall(void);

*// timer.c*

void timerinit(void);

*// trap.c*

void idtinit(void);

extern uint ticks;

void tvinit(void);

extern struct spinlock tickslock;

*// uart.c*

void uartinit(void);

void uartintr(void);

void uartputc(int);

*// vm.c*

void seginit(void);

void kvmalloc(void);

pde\_t\* setupkvm(void);

char\* uva2ka(pde\_t\*, char\*);

int allocuvm(pde\_t\*, uint, uint);

int deallocuvm(pde\_t\*, uint, uint);

void freevm(pde\_t\*);

void inituvm(pde\_t\*, char\*, uint);

int loaduvm(pde\_t\*, char\*, struct inode\*, uint, uint);

pde\_t\* copyuvm(pde\_t\*, uint);

void switchuvm(struct proc\*);

void switchkvm(void);

int copyout(pde\_t\*, uint, void\*, uint);

void clearpteu(pde\_t \**pgdir*, char \**uva*);

extern struct physframe\_info pf\_info[];

extern const int PFN\_TABLE\_SIZE;

struct vref;

int sw\_vtop(pde\_t \**pgdir*, const void \**va*, uint \**pa\_out*, uint \**flags\_out*);

void ipt\_init(void);

void ipt\_insert(uint *pfn*, int *pid*, uint *va\_page*, uint *flags*);

int ipt\_remove(int *pid*, uint *va\_page*, uint *pfn*);

int ipt\_query(uint *pfn*, struct vref \**kbuf*, int *max*); *// 커널버퍼에 채워줌*

void stlb\_init(void);

int stlb\_lookup(int *pid*, uint *va\_page*, uint \**pa\_page*, uint \**flags*);

void stlb\_fill(int *pid*, uint *va\_page*, uint *pa\_page*, uint *flags*);

void stlb\_stats(uint \**hits*, uint \**misses*);

void ipt\_purge\_pid(int *pid*);

int ipt\_update\_flags(int *pid*, uint *va\_page*, uint *pfn*, uint *newflags*);

void stlb\_invalidate\_va(int *pid*, uint *va\_page*);

void stlb\_update\_flags(int *pid*, uint *va\_page*, uint *newflags*);

void stlb\_purge\_pid(int *pid*);

int cow\_fault(pde\_t \**pgdir*, uint *va*);

void pf\_mark\_ready(void);

extern int pf\_ready;

*// number of elements in fixed-size array*

#define NELEM(*x*) (sizeof(x)/sizeof((x)[0]))

<vm.c>

#include "param.h"

#include "types.h"

#include "defs.h"

#include "x86.h"

#include "memlayout.h"

#include "mmu.h"

#include "proc.h"

#include "elf.h"

#include "spinlock.h"

\_\_attribute\_\_((weak)) void pf\_reset(uint *pfn*) { */\* no-op; 실제 구현 있으면 그것이 링크됨 \*/* }

int pf\_ready = 0; *// pf 서브시스템 준비 여부 플래그 (전역 정의)*

void pf\_mark\_ready(void) { *// 부팅 시 1로 세팅*

pf\_ready = 1;

}

extern volatile uint \*lapic; *// lapic.c에 정의됨*

static inline int safe\_curpid(void) {

*// mpinit() 이전엔 lapic이 0 → 절대 myproc()을 부르지 말고 -1(커널)로*

if(lapic == 0) return -1;

struct proc \*p = myproc();

return p ? p->pid : -1;

}

*// ---------- (A) Software Page Walker ----------*

int

sw\_vtop(pde\_t \**pgdir*, const void \**va*, uint \**pa\_out*, uint \**flags\_out*)

{

uint a = (uint)va;

pde\_t pde = pgdir[PDX(a)];

if(!(pde & PTE\_P)) return -1;

pte\_t \*pgtab = (pte\_t\*)P2V(PTE\_ADDR(pde));

pte\_t pte = pgtab[PTX(a)];

if(!(pte & PTE\_P)) return -1;

uint pa\_page = PTE\_ADDR(pte);

uint pa = pa\_page | (a & 0xFFF);

if(pa\_out) \*pa\_out = pa;

if(flags\_out) \*flags\_out = PTE\_FLAGS(pte);

return 0;

}

*// ---------- (B) Soft TLB (direct-mapped) ----------*

#define STLB\_BITS 9

#define STLB\_SIZE (1<<STLB\_BITS)

struct stlb\_entry { uint pid, va\_page, pa\_page, flags; };

static struct stlb\_entry stlb[STLB\_SIZE];

static struct spinlock stlblk;

static uint tlb\_hits, tlb\_misses;

static inline uint stlb\_idx(uint *pid*, uint *va\_page*){

return (pid\*1315423911u ^ va\_page) & (STLB\_SIZE-1);

}

void stlb\_init(void){

initlock(&stlblk, "stlb");

memset(stlb, 0, sizeof(stlb));

tlb\_hits = tlb\_misses = 0;

}

int stlb\_lookup(int *pid*, uint *va\_page*, uint \**pa\_page*, uint \**flags*){

acquire(&stlblk);

uint i = stlb\_idx(pid, va\_page);

if(stlb[i].pid==pid && stlb[i].va\_page==va\_page){

if(pa\_page) \*pa\_page = stlb[i].pa\_page;

if(flags) \*flags = stlb[i].flags;

tlb\_hits++;

release(&stlblk);

return 1;

}

tlb\_misses++;

release(&stlblk);

return 0;

}

void stlb\_fill(int *pid*, uint *va\_page*, uint *pa\_page*, uint *flags*){

acquire(&stlblk);

uint i = stlb\_idx(pid, va\_page);

stlb[i].pid = pid; stlb[i].va\_page = va\_page;

stlb[i].pa\_page = pa\_page; stlb[i].flags = flags;

release(&stlblk);

}

void stlb\_stats(uint \**hits*, uint \**misses*){

acquire(&stlblk);

if(hits) \*hits = tlb\_hits;

if(misses) \*misses = tlb\_misses;

release(&stlblk);

}

void stlb\_invalidate\_va(int *pid*, uint *va\_page*){

acquire(&stlblk);

uint i = stlb\_idx(pid, va\_page);

if(stlb[i].pid==pid && stlb[i].va\_page==va\_page){

stlb[i].pid=0; stlb[i].va\_page=0; stlb[i].pa\_page=0; stlb[i].flags=0;

}

release(&stlblk);

}

void stlb\_update\_flags(int *pid*, uint *va\_page*, uint *newflags*){

acquire(&stlblk);

uint i = stlb\_idx(pid, va\_page);

if(stlb[i].pid==pid && stlb[i].va\_page==va\_page)

stlb[i].flags = newflags;

release(&stlblk);

}

void stlb\_purge\_pid(int *pid*){

acquire(&stlblk);

for(int i=0;i<STLB\_SIZE;i++)

if(stlb[i].pid==pid)

stlb[i].pid=0, stlb[i].va\_page=0, stlb[i].pa\_page=0, stlb[i].flags=0;

release(&stlblk);

}

*// ---------- (C) IPT: 해시 + 풀알로케이터 ----------*

#define IPT\_BUCKETS 8192

struct ipt\_entry{

uint pfn; *// 물리 프레임 번호*

uint pid; *// 소유 PID (-1: 커널)*

uint va; *// 페이지 기준 VA*

uint flags; *// PTE 권한 스냅샷*

uint refcnt; *// (옵션)*

struct ipt\_entry \*next;

};

static struct ipt\_entry \*ipt\_hash[IPT\_BUCKETS];

static struct spinlock iptlk;

#define IPT\_POOL\_SIZE 65536

static struct ipt\_entry ipt\_pool[IPT\_POOL\_SIZE];

static struct ipt\_entry \*ipt\_free;

static inline uint ipt\_h(uint *pfn*){ return pfn & (IPT\_BUCKETS-1); }

static int ipt\_ready = 0; *// mpinit() 이후 ipt\_init()이 켜줌*

void ipt\_init(void){

initlock(&iptlk, "ipt");

memset(ipt\_hash, 0, sizeof(ipt\_hash));

for(int i=0;i<IPT\_POOL\_SIZE-1;i++) ipt\_pool[i].next = &ipt\_pool[i+1];

ipt\_pool[IPT\_POOL\_SIZE-1].next = 0;

ipt\_free = &ipt\_pool[0];

ipt\_ready = 1; *// 여기서 활성화*

}

static struct ipt\_entry\* ipt\_alloc\_ent(void){

if(!ipt\_free) return 0;

struct ipt\_entry \*e = ipt\_free;

ipt\_free = e->next;

return e;

}

static void ipt\_free\_ent(struct ipt\_entry \**e*){

e->next = ipt\_free; ipt\_free = e;

}

void ipt\_insert(uint *pfn*, int *pid*, uint *va\_page*, uint *flags*){

if(!ipt\_ready) return; *// 부팅 초기에는 그냥 스킵*

acquire(&iptlk);

struct ipt\_entry \*e = ipt\_alloc\_ent();

if(e){

e->pfn=pfn; e->pid=pid; e->va=va\_page; e->flags=flags; e->refcnt=1;

uint b = ipt\_h(pfn);

e->next = ipt\_hash[b];

ipt\_hash[b] = e;

}

release(&iptlk);

}

*// --- vm.c: IPT에서 (pid, va\_page, pfn) 하나 제거 ---*

*// PTE/물리프레임은 절대 건드리지 않는다(프레임 해제는 deallocuvm가 담당).*

*// va\_page는 반드시 PGROUNDDOWN 한 값으로 비교.*

int

ipt\_remove(int *pid*, uint *va\_page*, uint *pfn*)

{

if (!ipt\_ready) return 0;

va\_page = PGROUNDDOWN(va\_page);

int removed = 0;

acquire(&iptlk);

uint b = ipt\_h(pfn);

struct ipt\_entry \*\*pp = &ipt\_hash[b], \*cur;

while ((cur = \*pp)) {

if (cur->pid == pid && cur->va == va\_page && cur->pfn == pfn) {

\*pp = cur->next; *// unlink*

*// 엔트리 메타만 free (물리 프레임 kfree 절대 금지)*

ipt\_free\_ent(cur);

removed = 1;

break;

}

pp = &cur->next;

}

release(&iptlk);

return removed;

}

int ipt\_query(uint *pfn*, struct vref \**kbuf*, int *max*){

if(!ipt\_ready) return 0; *// 준비 전이면 결과 없음*

int n=0;

acquire(&iptlk);

struct ipt\_entry \*e;

for (e = ipt\_hash[ipt\_h(pfn)]; e && n < max; e = e->next) {

if (e->pfn != pfn) continue;

kbuf[n].pid = e->pid;

kbuf[n].va = e->va;

kbuf[n].flags = e->flags;

n++;

}

release(&iptlk);

return n;

}

int ipt\_update\_flags(int *pid*, uint *va\_page*, uint *pfn*, uint *newflags*){

if(!ipt\_ready) return 0;

int updated = 0;

acquire(&iptlk);

struct ipt\_entry \*e;

for (e = ipt\_hash[ipt\_h(pfn)]; e; e = e->next) {

if (e->pid == pid && e->va == va\_page && e->pfn == pfn) {

e->flags = newflags;

updated = 1;

break;

}

}

release(&iptlk);

return updated;

}

void

ipt\_purge\_pid(int *pid*)

{

if (!ipt\_ready) return;

acquire(&iptlk);

for (int b = 0; b < IPT\_BUCKETS; b++) {

struct ipt\_entry \*\*pp = &ipt\_hash[b], \*e;

while ((e = \*pp)) {

if (e->pid == pid) {

\*pp = e->next; *// unlink*

ipt\_free\_ent(e); *// 메타만 free*

continue; *// 다음 노드 검사(건너뛰기 방지)*

}

pp = &e->next;

}

}

release(&iptlk);

}

*// pfn(= 물리 프레임 번호) 의 현재 IPT 참조 개수 반환*

int ipt\_refcount(uint *pfn*) {

struct vref tmp[64];

int total = 0;

*// ipt\_query(pfn, buf, max) 는 최대 'max'개를 채워주고 실제 개수를 리턴한다고 가정*

int n = ipt\_query(pfn, tmp, 64);

if(n > 0) total += n;

return total;

}

*//----------------------------------------------------------------------*

extern char data[]; *// defined by kernel.ld*

pde\_t \*kpgdir; *// for use in scheduler()*

*// Set up CPU's kernel segment descriptors.*

*// Run once on entry on each CPU.*

void

seginit(void)

{

struct cpu \*c;

*// Map "logical" addresses to virtual addresses using identity map.*

*// Cannot share a CODE descriptor for both kernel and user*

*// because it would have to have DPL\_USR, but the CPU forbids*

*// an interrupt from CPL=0 to DPL=3.*

c = &cpus[cpuid()];

c->gdt[SEG\_KCODE] = SEG(STA\_X|STA\_R, 0, 0xffffffff, 0);

c->gdt[SEG\_KDATA] = SEG(STA\_W, 0, 0xffffffff, 0);

c->gdt[SEG\_UCODE] = SEG(STA\_X|STA\_R, 0, 0xffffffff, DPL\_USER);

c->gdt[SEG\_UDATA] = SEG(STA\_W, 0, 0xffffffff, DPL\_USER);

lgdt(c->gdt, sizeof(c->gdt));

}

*// Return the address of the PTE in page table pgdir*

*// that corresponds to virtual address va. If alloc!=0,*

*// create any required page table pages.*

static pte\_t \*

walkpgdir(pde\_t \**pgdir*, const void \**va*, int *alloc*)

{

pde\_t \*pde;

pte\_t \*pgtab;

pde = &pgdir[PDX(va)];

if(\*pde & PTE\_P){

pgtab = (pte\_t\*)P2V(PTE\_ADDR(\*pde));

} else {

if(!alloc || (pgtab = (pte\_t\*)kalloc()) == 0)

return 0;

*// Make sure all those PTE\_P bits are zero.*

memset(pgtab, 0, PGSIZE);

*// The permissions here are overly generous, but they can*

*// be further restricted by the permissions in the page table*

*// entries, if necessary.*

\*pde = V2P(pgtab) | PTE\_P | PTE\_W | PTE\_U;

}

return &pgtab[PTX(va)];

}

*// Create PTEs for virtual addresses starting at va that refer to*

*// physical addresses starting at pa. va and size might not*

*// be page-aligned.*

*// ---------- (D) vm.c의 기존 경로에 IPT/SoftTLB 연동 ----------*

*// mappages(): 매핑 생성 시 IPT에 삽입*

int

mappages(pde\_t \**pgdir*, void \**va*, uint *size*, uint *pa*, int *perm*)

{

char \*a, \*last;

pte\_t \*pte;

a = (char\*)PGROUNDDOWN((uint)va);

last = (char\*)PGROUNDDOWN(((uint)va) + size - 1);

for(;;){

if((pte = walkpgdir(pgdir, a, 1)) == 0)

return -1;

if(\*pte & PTE\_P)

panic("remap");

\*pte = pa | perm | PTE\_P;

*// ---- IPT 삽입 (유저 매핑만) ----*

uint flags = PTE\_FLAGS(\*pte);

if (flags & PTE\_U) { *// 커널 매핑(PTE\_U==0)은 건너뜀*

int pid = safe\_curpid(); *// 부팅 초기엔 -1이 나오도록 이미 안전*

uint va\_page = (uint)a; *// a는 이미 PGROUNDDOWN 된 값*

uint pfn = (pa >> 12);

ipt\_insert(pfn, pid, va\_page, flags);

}

if(a == last) break;

a += PGSIZE;

pa += PGSIZE;

}

return 0;

}

*// There is one page table per process, plus one that's used when*

*// a CPU is not running any process (kpgdir). The kernel uses the*

*// current process's page table during system calls and interrupts;*

*// page protection bits prevent user code from using the kernel's*

*// mappings.*

*//*

*// setupkvm() and exec() set up every page table like this:*

*//*

*// 0..KERNBASE: user memory (text+data+stack+heap), mapped to*

*// phys memory allocated by the kernel*

*// KERNBASE..KERNBASE+EXTMEM: mapped to 0..EXTMEM (for I/O space)*

*// KERNBASE+EXTMEM..data: mapped to EXTMEM..V2P(data)*

*// for the kernel's instructions and r/o data*

*// data..KERNBASE+PHYSTOP: mapped to V2P(data)..PHYSTOP,*

*// rw data + free physical memory*

*// 0xfe000000..0: mapped direct (devices such as ioapic)*

*//*

*// The kernel allocates physical memory for its heap and for user memory*

*// between V2P(end) and the end of physical memory (PHYSTOP)*

*// (directly addressable from end..P2V(PHYSTOP)).*

*// This table defines the kernel's mappings, which are present in*

*// every process's page table.*

static struct kmap {

void \*virt;

uint phys\_start;

uint phys\_end;

int perm;

} kmap[] = {

{ (void\*)KERNBASE, 0, EXTMEM, PTE\_W}, *// I/O space*

{ (void\*)KERNLINK, V2P(KERNLINK), V2P(data), 0}, *// kern text+rodata*

{ (void\*)data, V2P(data), PHYSTOP, PTE\_W}, *// kern data+memory*

{ (void\*)DEVSPACE, DEVSPACE, 0, PTE\_W}, *// more devices*

};

*// Set up kernel part of a page table.*

pde\_t\*

setupkvm(void)

{

pde\_t \*pgdir;

struct kmap \*k;

if((pgdir = (pde\_t\*)kalloc()) == 0)

return 0;

memset(pgdir, 0, PGSIZE);

if (P2V(PHYSTOP) > (void\*)DEVSPACE)

panic("PHYSTOP too high");

for(k = kmap; k < &kmap[NELEM(kmap)]; k++)

if(mappages(pgdir, k->virt, k->phys\_end - k->phys\_start,

(uint)k->phys\_start, k->perm) < 0) {

freevm(pgdir);

return 0;

}

return pgdir;

}

*// Allocate one page table for the machine for the kernel address*

*// space for scheduler processes.*

void

kvmalloc(void)

{

kpgdir = setupkvm();

switchkvm();

extern void stlb\_init(void);

extern void ipt\_init(void);

static int inited = 0;

if (!inited) {

stlb\_init();

ipt\_init();

pf\_mark\_ready(); *// 이 시점 이후에만 kfree가 pf\_reset 수행*

inited = 1;

}

}

*// Switch h/w page table register to the kernel-only page table,*

*// for when no process is running.*

void

switchkvm(void)

{

lcr3(V2P(kpgdir)); *// switch to the kernel page table*

}

*// Switch TSS and h/w page table to correspond to process p.*

void

switchuvm(struct proc \**p*)

{

if(p == 0)

panic("switchuvm: no process");

if(p->kstack == 0)

panic("switchuvm: no kstack");

if(p->pgdir == 0)

panic("switchuvm: no pgdir");

pushcli();

mycpu()->gdt[SEG\_TSS] = SEG16(STS\_T32A, &mycpu()->ts,

sizeof(mycpu()->ts)-1, 0);

mycpu()->gdt[SEG\_TSS].s = 0;

mycpu()->ts.ss0 = SEG\_KDATA << 3;

mycpu()->ts.esp0 = (uint)p->kstack + KSTACKSIZE;

*// setting IOPL=0 in eflags \*and\* iomb beyond the tss segment limit*

*// forbids I/O instructions (e.g., inb and outb) from user space*

mycpu()->ts.iomb = (ushort) 0xFFFF;

ltr(SEG\_TSS << 3);

lcr3(V2P(p->pgdir)); *// switch to process's address space*

popcli();

}

*// Load the initcode into address 0 of pgdir.*

*// sz must be less than a page.*

void

inituvm(pde\_t \**pgdir*, char \**init*, uint *sz*)

{

char \*mem;

if(sz >= PGSIZE)

panic("inituvm: more than a page");

mem = kalloc();

memset(mem, 0, PGSIZE);

mappages(pgdir, 0, PGSIZE, V2P(mem), PTE\_W|PTE\_U);

memmove(mem, init, sz);

}

*// Load a program segment into pgdir. addr must be page-aligned*

*// and the pages from addr to addr+sz must already be mapped.*

int

loaduvm(pde\_t \**pgdir*, char \**addr*, struct inode \**ip*, uint *offset*, uint *sz*)

{

uint i, pa, n;

pte\_t \*pte;

if((uint) addr % PGSIZE != 0)

panic("loaduvm: addr must be page aligned");

for(i = 0; i < sz; i += PGSIZE){

if((pte = walkpgdir(pgdir, addr+i, 0)) == 0)

panic("loaduvm: address should exist");

pa = PTE\_ADDR(\*pte);

if(sz - i < PGSIZE)

n = sz - i;

else

n = PGSIZE;

if(readi(ip, P2V(pa), offset+i, n) != n)

return -1;

}

return 0;

}

*// Allocate page tables and physical memory to grow process from oldsz to*

*// newsz, which need not be page aligned. Returns new size or 0 on error.*

int

allocuvm(pde\_t \**pgdir*, uint *oldsz*, uint *newsz*)

{

char \*mem;

uint a;

if(newsz >= KERNBASE)

return 0;

if(newsz < oldsz)

return oldsz;

a = PGROUNDUP(oldsz);

for(; a < newsz; a += PGSIZE){

mem = kalloc();

if(mem == 0){

cprintf("allocuvm out of memory\n");

deallocuvm(pgdir, newsz, oldsz);

return 0;

}

memset(mem, 0, PGSIZE);

if(mappages(pgdir, (char\*)a, PGSIZE, V2P(mem), PTE\_W|PTE\_U) < 0){

cprintf("allocuvm out of memory (2)\n");

deallocuvm(pgdir, newsz, oldsz);

kfree(mem);

return 0;

}

}

return newsz;

}

*// Deallocate user pages to bring the process size from oldsz to*

*// newsz. oldsz and newsz need not be page-aligned, nor does newsz*

*// need to be less than oldsz. oldsz can be larger than the actual*

*// process size. Returns the new process size.*

*// vm.c*

int

deallocuvm(pde\_t \**pgdir*, uint *oldsz*, uint *newsz*)

{

pte\_t \*pte;

uint a, pa;

if (newsz >= oldsz)

return oldsz;

a = PGROUNDUP(newsz);

for (; a < oldsz; a += PGSIZE) {

pte = walkpgdir(pgdir, (char\*)a, 0);

if (!pte) { a = PGADDR(PDX(a)+1,0,0) - PGSIZE; continue; }

if ((\*pte & PTE\_P) == 0) continue;

pa = PTE\_ADDR(\*pte);

if (pa == 0) panic("deallocuvm kfree");

uint va\_page = PGROUNDDOWN(a);

uint pfn = pa >> 12;

*// 현재 실행중인 주소공간인지 확인 (부모가 자식 pgdir을 free하는 경우가 많음)*

int is\_cur\_pgdir = (myproc() && myproc()->pgdir == pgdir);

int pid\_cur = safe\_curpid(); *// 현재 프로세스 pid (is\_cur\_pgdir일 때만 사용)*

*// 1) 먼저 PTE를 죽인다*

\*pte = 0;

*// 2) 현재 주소공간일 때만 HW TLB flush 및 STLB/IPT 조작*

if (is\_cur\_pgdir) {

lcr3(V2P(pgdir)); *// HW TLB flush*

stlb\_invalidate\_va(pid\_cur, va\_page);

ipt\_remove(pid\_cur, va\_page, pfn); *// 부모가 자식 pgdir을 free할 땐 실행하지 않음*

}

*// 3) 남은 레퍼런스가 없을 때만 프레임 해제*

if (ipt\_refcount(pfn) == 0) {

pf\_reset(pfn); *// no-op이면 무시*

kfree(P2V(pa));

}

}

return newsz;

}

*// Free a page table and all the physical memory pages*

*// in the user part.*

void

freevm(pde\_t \**pgdir*)

{

uint i;

if(pgdir == 0)

panic("freevm: no pgdir");

deallocuvm(pgdir, KERNBASE, 0);

for(i = 0; i < NPDENTRIES; i++){

if(pgdir[i] & PTE\_P){

char \* v = P2V(PTE\_ADDR(pgdir[i]));

kfree(v);

}

}

kfree((char\*)pgdir);

}

*// Clear PTE\_U on a page. Used to create an inaccessible*

*// page beneath the user stack.*

void

clearpteu(pde\_t \**pgdir*, char \**uva*)

{

pte\_t \*pte;

pte = walkpgdir(pgdir, uva, 0);

if(pte == 0)

panic("clearpteu");

\*pte &= ~PTE\_U;

*// IPT flags 동기화*

int pid = safe\_curpid();

uint pa\_page = PTE\_ADDR(\*pte);

ipt\_update\_flags(pid, PGROUNDDOWN((uint)uva), pa\_page>>12, PTE\_FLAGS(\*pte));

stlb\_update\_flags(pid, PGROUNDDOWN((uint)uva), PTE\_FLAGS(\*pte)); *// 소프트 TLB 동기화*

}

*// Given a parent process's page table, create a copy*

*// of it for a child.*

pde\_t\*

copyuvm(pde\_t \**pgdir*, uint *sz*)

{

pde\_t \*d;

pte\_t \*pte;

uint pa, i, flags;

char \*mem;

if((d = setupkvm()) == 0)

return 0;

for(i = 0; i < sz; i += PGSIZE){

if((pte = walkpgdir(pgdir, (void \*) i, 0)) == 0)

panic("copyuvm: pte should exist");

if(!(\*pte & PTE\_P))

panic("copyuvm: page not present");

pa = PTE\_ADDR(\*pte);

flags = PTE\_FLAGS(\*pte);

if((mem = kalloc()) == 0)

goto bad;

memmove(mem, (char\*)P2V(pa), PGSIZE);

if(mappages(d, (void\*)i, PGSIZE, V2P(mem), flags) < 0) {

kfree(mem);

goto bad;

}

}

return d;

bad:

freevm(d);

return 0;

}

*// 부모 pgdir에서 va의 PTE를 찾아 동일 pa를 자식 pgdir에 공유 매핑.*

*// 두 쪽 모두 PTE\_W를 내리고 PTE\_COW를 세팅한다.*

static int

cow\_share\_page(pde\_t \**pgdir\_parent*, pde\_t \**pgdir\_child*, uint *va*, int *child\_pid*)

{

pte\_t \*ppte = walkpgdir(pgdir\_parent, (char\*)PGROUNDDOWN(va), 0);

if(ppte == 0 || (\*ppte & PTE\_P) == 0 || (\*ppte & PTE\_U) == 0)

return -1;

uint pa = PTE\_ADDR(\*ppte);

uint flags = PTE\_FLAGS(\*ppte);

uint va\_page = PGROUNDDOWN(va);

uint pfn = pa >> 12;

*// 1) 부모 PTE를 RO + COW로*

uint cow\_flags = (flags & ~PTE\_W) | PTE\_COW;

\*ppte = pa | cow\_flags;

*// 1-1) (실행중이 부모 컨텍스트라면) 하드웨어 TLB도 무효화*

stlb\_invalidate\_va(safe\_curpid(), va\_page); *// 소프트TLB 사용*

*// 2) IPT: 부모 엔트리 갱신 (없으면 새로 생성)*

int ppid = safe\_curpid(); *// 부모 pid*

int updated = ipt\_update\_flags(ppid, va\_page, pfn, cow\_flags);

if(updated == 0){

*// 부모 엔트리가 없었음 → 부모 것도 만들어 둔다*

ipt\_insert(pfn, ppid, va\_page, cow\_flags);

}

*// 3) 자식 PTE를 직접 만든다(== mappages() 사용 금지!)*

pte\_t \*cpte = walkpgdir(pgdir\_child, (char\*)va\_page, 1); *// 필요 시 PT 생성*

if(cpte == 0) return -1;

\*cpte = pa | cow\_flags; *// 동일 PFN, RO+COW*

*// 4) IPT: 자식쪽 삽입*

ipt\_insert(pfn, child\_pid, va\_page, cow\_flags);

return 0;

}

*// 기존 copyuvm(실제 복사) 대신: 모든 유저 페이지를 COW 공유로 붙인다.*

pde\_t\*

cowuvm(pde\_t \**pgdir\_parent*, uint *sz*, int *child\_pid*)

{

pde\_t \*d;

if((d = setupkvm()) == 0)

return 0;

for(uint va = 0; va < sz; va += PGSIZE){

pte\_t \*pte = walkpgdir(pgdir\_parent, (char\*)va, 0);

if(pte == 0) { va = PGADDR(PDX(va)+1, 0, 0) - PGSIZE; continue; }

if((\*pte & PTE\_P) == 0) continue; *// unmapped*

if((\*pte & PTE\_U) == 0) continue; *// 커널 전용은 공유 대상 아님*

if(cow\_share\_page(pgdir\_parent, d, va, child\_pid) < 0){ freevm(d); return 0; }

}

*// 부모의 TLB를 한번에 싹 비움(부모 PTE가 바뀌었기 때문)*

lcr3(V2P(pgdir\_parent));

return d;

}

*// faulting va에 대해 COW 처리: 새 페이지를 만들어 내용 복사 후 쓰기 가능하게 갱신*

int cow\_fault(pde\_t \**pgdir*, uint *va*)

{

uint va\_page = PGROUNDDOWN(va);

pte\_t \*pte = walkpgdir(pgdir, (char\*)va\_page, 0);

if(!pte || (\*pte & PTE\_P) == 0) return -1;

if((\*pte & PTE\_COW) == 0) return -1;

uint old\_pa = PTE\_ADDR(\*pte);

uint old\_flag = PTE\_FLAGS(\*pte);

char \*mem = kalloc();

if(mem == 0) return -1;

memmove(mem, (char\*)P2V(old\_pa), PGSIZE);

int pid = safe\_curpid();

*// STLB/ipt에서 기존 매핑 제거*

stlb\_invalidate\_va(pid, va\_page);

ipt\_remove(pid, va\_page, old\_pa >> 12);

*// 새 페이지로 재매핑: 쓰기 가능, COW 해제*

\*pte = V2P(mem) | ((old\_flag | PTE\_W) & ~PTE\_COW);

uint nflags = PTE\_FLAGS(\*pte);

*// IPT에 새 프레임 삽입*

ipt\_insert((V2P(mem) >> 12), pid, va\_page, nflags);

*// 하드웨어 TLB flush*

lcr3(V2P(pgdir));

return 0;

}

*//PAGEBREAK!*

*// Map user virtual address to kernel address.*

char\*

uva2ka(pde\_t \**pgdir*, char \**uva*)

{

pte\_t \*pte;

pte = walkpgdir(pgdir, uva, 0);

if((\*pte & PTE\_P) == 0)

return 0;

if((\*pte & PTE\_U) == 0)

return 0;

return (char\*)P2V(PTE\_ADDR(\*pte));

}

*// Copy len bytes from p to user address va in page table pgdir.*

*// Most useful when pgdir is not the current page table.*

*// uva2ka ensures this only works for PTE\_U pages.*

int

copyout(pde\_t \**pgdir*, uint *va*, void \**p*, uint *len*)

{

char \*buf, \*pa0;

uint n, va0;

buf = (char\*)p;

while(len > 0){

va0 = (uint)PGROUNDDOWN(va);

pa0 = uva2ka(pgdir, (char\*)va0);

if(pa0 == 0)

return -1;

n = PGSIZE - (va - va0);

if(n > len)

n = len;

memmove(pa0 + (va - va0), buf, n);

len -= n;

buf += n;

va = va0 + PGSIZE;

}

return 0;

}

*//PAGEBREAK!*

*// Blank page.*

*//PAGEBREAK!*

*// Blank page.*

*//PAGEBREAK!*

*// Blank page.*

<proc.c>

#include "types.h"

#include "defs.h"

#include "param.h"

#include "memlayout.h"

#include "mmu.h"

#include "x86.h"

#include "proc.h"

#include "spinlock.h"

struct {

struct spinlock lock;

struct proc proc[NPROC];

} ptable;

static struct proc \*initproc;

int nextpid = 1;

extern void forkret(void);

extern void trapret(void);

static void wakeup1(void \**chan*);

void

pinit(void)

{

initlock(&ptable.lock, "ptable");

}

*// Must be called with interrupts disabled*

int

cpuid() {

return mycpu()-cpus;

}

*// Must be called with interrupts disabled to avoid the caller being*

*// rescheduled between reading lapicid and running through the loop.*

struct cpu\*

mycpu(void)

{

int apicid, i;

if(readeflags()&FL\_IF)

panic("mycpu called with interrupts enabled\n");

apicid = lapicid();

*// APIC IDs are not guaranteed to be contiguous. Maybe we should have*

*// a reverse map, or reserve a register to store &cpus[i].*

for (i = 0; i < ncpu; ++i) {

if (cpus[i].apicid == apicid)

return &cpus[i];

}

panic("unknown apicid\n");

}

*// Disable interrupts so that we are not rescheduled*

*// while reading proc from the cpu structure*

struct proc\*

myproc(void) {

struct cpu \*c;

struct proc \*p;

pushcli();

c = mycpu();

p = c->proc;

popcli();

return p;

}

*//PAGEBREAK: 32*

*// Look in the process table for an UNUSED proc.*

*// If found, change state to EMBRYO and initialize*

*// state required to run in the kernel.*

*// Otherwise return 0.*

static struct proc\*

allocproc(void)

{

struct proc \*p;

char \*sp;

acquire(&ptable.lock);

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)

if(p->state == UNUSED)

goto found;

release(&ptable.lock);

return 0;

found:

p->state = EMBRYO;

p->pid = nextpid++;

release(&ptable.lock);

*// Allocate kernel stack.*

if((p->kstack = kalloc()) == 0){

p->state = UNUSED;

return 0;

}

sp = p->kstack + KSTACKSIZE;

*// Leave room for trap frame.*

sp -= sizeof \*p->tf;

p->tf = (struct trapframe\*)sp;

*// Set up new context to start executing at forkret,*

*// which returns to trapret.*

sp -= 4;

\*(uint\*)sp = (uint)trapret;

sp -= sizeof \*p->context;

p->context = (struct context\*)sp;

memset(p->context, 0, sizeof \*p->context);

p->context->eip = (uint)forkret;

return p;

}

*//PAGEBREAK: 32*

*// Set up first user process.*

void

userinit(void)

{

struct proc \*p;

extern char \_binary\_initcode\_start[], \_binary\_initcode\_size[];

p = allocproc();

initproc = p;

if((p->pgdir = setupkvm()) == 0)

panic("userinit: out of memory?");

inituvm(p->pgdir, \_binary\_initcode\_start, (int)\_binary\_initcode\_size);

p->sz = PGSIZE;

memset(p->tf, 0, sizeof(\*p->tf));

p->tf->cs = (SEG\_UCODE << 3) | DPL\_USER;

p->tf->ds = (SEG\_UDATA << 3) | DPL\_USER;

p->tf->es = p->tf->ds;

p->tf->ss = p->tf->ds;

p->tf->eflags = FL\_IF;

p->tf->esp = PGSIZE;

p->tf->eip = 0; *// beginning of initcode.S*

safestrcpy(p->name, "initcode", sizeof(p->name));

p->cwd = namei("/");

*// this assignment to p->state lets other cores*

*// run this process. the acquire forces the above*

*// writes to be visible, and the lock is also needed*

*// because the assignment might not be atomic.*

acquire(&ptable.lock);

p->state = RUNNABLE;

release(&ptable.lock);

}

*// Grow current process's memory by n bytes.*

*// Return 0 on success, -1 on failure.*

int

growproc(int *n*)

{

uint sz;

struct proc \*curproc = myproc();

sz = curproc->sz;

if(n > 0){

if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0)

return -1;

} else if(n < 0){

if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0)

return -1;

}

curproc->sz = sz;

switchuvm(curproc);

return 0;

}

*// Create a new process copying p as the parent.*

*// Sets up stack to return as if from system call.*

*// Caller must set state of returned proc to RUNNABLE.*

int

fork(void)

{

int i, pid;

struct proc \*np;

struct proc \*curproc = myproc();

*// Allocate process.*

if((np = allocproc()) == 0){

return -1;

}

if((np->pgdir = cowuvm(curproc->pgdir, curproc->sz, np->pid)) == 0){

kfree(np->kstack); np->kstack = 0; np->state = UNUSED; release(&ptable.lock);

return -1;

}

np->sz = curproc->sz;

np->parent = curproc;

\*np->tf = \*curproc->tf;

*// Clear %eax so that fork returns 0 in the child.*

np->tf->eax = 0;

for(i = 0; i < NOFILE; i++)

if(curproc->ofile[i])

np->ofile[i] = filedup(curproc->ofile[i]);

np->cwd = idup(curproc->cwd);

safestrcpy(np->name, curproc->name, sizeof(curproc->name));

pid = np->pid;

acquire(&ptable.lock);

np->state = RUNNABLE;

release(&ptable.lock);

return pid;

}

*// Exit the current process. Does not return.*

*// An exited process remains in the zombie state*

*// until its parent calls wait() to find out it exited.*

void

exit(void)

{

struct proc \*curproc = myproc();

struct proc \*p;

int fd;

if(curproc == initproc)

panic("init exiting");

*// Close all open files.*

for(fd = 0; fd < NOFILE; fd++){

if(curproc->ofile[fd]){

fileclose(curproc->ofile[fd]);

curproc->ofile[fd] = 0;

}

}

begin\_op();

iput(curproc->cwd);

end\_op();

curproc->cwd = 0;

*// 이 PID는 이제 죽을 것이므로 소프트 TLB에서 싹 지움*

ipt\_purge\_pid(curproc->pid); *// IPT 비움*

stlb\_purge\_pid(curproc->pid);

acquire(&ptable.lock);

*// Parent might be sleeping in wait().*

wakeup1(curproc->parent);

*// Pass abandoned children to init.*

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){

if(p->parent == curproc){

p->parent = initproc;

if(p->state == ZOMBIE)

wakeup1(initproc);

}

}

*// Jump into the scheduler, never to return.*

curproc->state = ZOMBIE;

sched();

panic("zombie exit");

}

*// Wait for a child process to exit and return its pid.*

*// Return -1 if this process has no children.*

int

wait(void)

{

struct proc \*p;

int havekids, pid;

struct proc \*cur = myproc();

acquire(&ptable.lock);

for(;;){

havekids = 0;

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){

if(p->parent != cur)

continue;

havekids = 1;

if(p->state == ZOMBIE){

*// --- 1) 자식 자원 스냅샷(락 안에서) & 프로세스 엔트리 분리 ---*

pid = p->pid;

*// 자식 자원 로컬에 백업*

char \*child\_kstack = p->kstack;

pde\_t \*child\_pgdir = p->pgdir;

*// 다른 CPU가 이 엔트리를 재사용하더라도*

*// 우리가 들고 있는 포인터로 안전하게 free 할 수 있도록*

*// 엔트리와 자원을 '분리(detach)'한다.*

p->kstack = 0;

p->pgdir = 0;

*// 프로세스 메타를 바로 정리해 재사용 가능 상태로 돌려둔다.*

p->pid = 0;

p->parent = 0;

p->name[0]= 0;

p->killed = 0;

p->state = UNUSED;

*// --- 2) 락 해제 후, 자식 PID의 캐시/테이블 정리 + 실제 free ---*

release(&ptable.lock);

*// (중복 호출 무해) 자식 PID의 소프트 TLB/IPT 전부 비우기*

stlb\_purge\_pid(pid);

ipt\_purge\_pid(pid);

*// 실제 메모리 해제 (락 밖에서)*

if(child\_kstack) kfree(child\_kstack);

if(child\_pgdir) freevm(child\_pgdir);

return pid;

}

}

if(!havekids || cur->killed){

release(&ptable.lock);

return -1;

}

sleep(cur, &ptable.lock);

}

}

*//PAGEBREAK: 42*

*// Per-CPU process scheduler.*

*// Each CPU calls scheduler() after setting itself up.*

*// Scheduler never returns. It loops, doing:*

*// - choose a process to run*

*// - swtch to start running that process*

*// - eventually that process transfers control*

*// via swtch back to the scheduler.*

void

scheduler(void)

{

struct proc \*p;

struct cpu \*c = mycpu();

c->proc = 0;

for(;;){

*// Enable interrupts on this processor.*

sti();

*// Loop over process table looking for process to run.*

acquire(&ptable.lock);

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){

if(p->state != RUNNABLE)

continue;

*// Switch to chosen process. It is the process's job*

*// to release ptable.lock and then reacquire it*

*// before jumping back to us.*

c->proc = p;

switchuvm(p);

p->state = RUNNING;

swtch(&(c->scheduler), p->context);

switchkvm();

*// Process is done running for now.*

*// It should have changed its p->state before coming back.*

c->proc = 0;

}

release(&ptable.lock);

}

}

*// Enter scheduler. Must hold only ptable.lock*

*// and have changed proc->state. Saves and restores*

*// intena because intena is a property of this*

*// kernel thread, not this CPU. It should*

*// be proc->intena and proc->ncli, but that would*

*// break in the few places where a lock is held but*

*// there's no process.*

void

sched(void)

{

int intena;

struct proc \*p = myproc();

if(!holding(&ptable.lock))

panic("sched ptable.lock");

if(mycpu()->ncli != 1)

panic("sched locks");

if(p->state == RUNNING)

panic("sched running");

if(readeflags()&FL\_IF)

panic("sched interruptible");

intena = mycpu()->intena;

swtch(&p->context, mycpu()->scheduler);

mycpu()->intena = intena;

}

*// Give up the CPU for one scheduling round.*

void

yield(void)

{

acquire(&ptable.lock); *//DOC: yieldlock*

myproc()->state = RUNNABLE;

sched();

release(&ptable.lock);

}

*// A fork child's very first scheduling by scheduler()*

*// will swtch here. "Return" to user space.*

void

forkret(void)

{

static int first = 1;

*// Still holding ptable.lock from scheduler.*

release(&ptable.lock);

if (first) {

*// Some initialization functions must be run in the context*

*// of a regular process (e.g., they call sleep), and thus cannot*

*// be run from main().*

first = 0;

iinit(ROOTDEV);

initlog(ROOTDEV);

}

*// Return to "caller", actually trapret (see allocproc).*

}

*// Atomically release lock and sleep on chan.*

*// Reacquires lock when awakened.*

void

sleep(void \**chan*, struct spinlock \**lk*)

{

struct proc \*p = myproc();

if(p == 0)

panic("sleep");

if(lk == 0)

panic("sleep without lk");

*// Must acquire ptable.lock in order to*

*// change p->state and then call sched.*

*// Once we hold ptable.lock, we can be*

*// guaranteed that we won't miss any wakeup*

*// (wakeup runs with ptable.lock locked),*

*// so it's okay to release lk.*

if(lk != &ptable.lock){ *//DOC: sleeplock0*

acquire(&ptable.lock); *//DOC: sleeplock1*

release(lk);

}

*// Go to sleep.*

p->chan = chan;

p->state = SLEEPING;

sched();

*// Tidy up.*

p->chan = 0;

*// Reacquire original lock.*

if(lk != &ptable.lock){ *//DOC: sleeplock2*

release(&ptable.lock);

acquire(lk);

}

}

*//PAGEBREAK!*

*// Wake up all processes sleeping on chan.*

*// The ptable lock must be held.*

static void

wakeup1(void \**chan*)

{

struct proc \*p;

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)

if(p->state == SLEEPING && p->chan == chan)

p->state = RUNNABLE;

}

*// Wake up all processes sleeping on chan.*

void

wakeup(void \**chan*)

{

acquire(&ptable.lock);

wakeup1(chan);

release(&ptable.lock);

}

*// Kill the process with the given pid.*

*// Process won't exit until it returns*

*// to user space (see trap in trap.c).*

int

kill(int *pid*)

{

struct proc \*p;

acquire(&ptable.lock);

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){

if(p->pid == pid){

p->killed = 1;

*// Wake process from sleep if necessary.*

if(p->state == SLEEPING)

p->state = RUNNABLE;

release(&ptable.lock);

return 0;

}

}

release(&ptable.lock);

return -1;

}

*//PAGEBREAK: 36*

*// Print a process listing to console. For debugging.*

*// Runs when user types ^P on console.*

*// No lock to avoid wedging a stuck machine further.*

void

procdump(void)

{

static char \*states[] = {

[UNUSED] "unused",

[EMBRYO] "embryo",

[SLEEPING] "sleep ",

[RUNNABLE] "runble",

[RUNNING] "run ",

[ZOMBIE] "zombie"

};

int i;

struct proc \*p;

char \*state;

uint pc[10];

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){

if(p->state == UNUSED)

continue;

if(p->state >= 0 && p->state < NELEM(states) && states[p->state])

state = states[p->state];

else

state = "???";

cprintf("%d %s %s", p->pid, state, p->name);

if(p->state == SLEEPING){

getcallerpcs((uint\*)p->context->ebp+2, pc);

for(i=0; i<10 && pc[i] != 0; i++)

cprintf(" %p", pc[i]);

}

cprintf("\n");

}

}

<exec.c>

#include "types.h"

#include "param.h"

#include "memlayout.h"

#include "mmu.h"

#include "proc.h"

#include "defs.h"

#include "x86.h"

#include "elf.h"

int

exec(char \**path*, char \*\**argv*)

{

char \*s, \*last;

int i, off;

uint argc, sz, sp, ustack[3+MAXARG+1];

struct elfhdr elf;

struct inode \*ip;

struct proghdr ph;

pde\_t \*pgdir, \*oldpgdir;

struct proc \*curproc = myproc();

begin\_op();

if((ip = namei(path)) == 0){

end\_op();

cprintf("exec: fail\n");

return -1;

}

ilock(ip);

pgdir = 0;

*// Check ELF header*

if(readi(ip, (char\*)&elf, 0, sizeof(elf)) != sizeof(elf))

goto bad;

if(elf.magic != ELF\_MAGIC)

goto bad;

if((pgdir = setupkvm()) == 0)

goto bad;

*// Load program into memory.*

sz = 0;

for(i=0, off=elf.phoff; i<elf.phnum; i++, off+=sizeof(ph)){

if(readi(ip, (char\*)&ph, off, sizeof(ph)) != sizeof(ph))

goto bad;

if(ph.type != ELF\_PROG\_LOAD)

continue;

if(ph.memsz < ph.filesz)

goto bad;

if(ph.vaddr + ph.memsz < ph.vaddr)

goto bad;

if((sz = allocuvm(pgdir, sz, ph.vaddr + ph.memsz)) == 0)

goto bad;

if(ph.vaddr % PGSIZE != 0)

goto bad;

if(loaduvm(pgdir, (char\*)ph.vaddr, ip, ph.off, ph.filesz) < 0)

goto bad;

}

iunlockput(ip);

end\_op();

ip = 0;

*// Allocate two pages at the next page boundary.*

*// Make the first inaccessible. Use the second as the user stack.*

sz = PGROUNDUP(sz);

if((sz = allocuvm(pgdir, sz, sz + 2\*PGSIZE)) == 0)

goto bad;

clearpteu(pgdir, (char\*)(sz - 2\*PGSIZE));

sp = sz;

*// Push argument strings, prepare rest of stack in ustack.*

for(argc = 0; argv[argc]; argc++) {

if(argc >= MAXARG)

goto bad;

sp = (sp - (strlen(argv[argc]) + 1)) & ~3;

if(copyout(pgdir, sp, argv[argc], strlen(argv[argc]) + 1) < 0)

goto bad;

ustack[3+argc] = sp;

}

ustack[3+argc] = 0;

ustack[0] = 0xffffffff; *// fake return PC*

ustack[1] = argc;

ustack[2] = sp - (argc+1)\*4; *// argv pointer*

sp -= (3+argc+1) \* 4;

if(copyout(pgdir, sp, ustack, (3+argc+1)\*4) < 0)

goto bad;

*// Save program name for debugging.*

for(last=s=path; \*s; s++)

if(\*s == '/')

last = s+1;

safestrcpy(curproc->name, last, sizeof(curproc->name));

*// Commit to the user image.*

oldpgdir = curproc->pgdir;

curproc->pgdir = pgdir;

curproc->sz = sz;

curproc->tf->eip = elf.entry; *// main*

curproc->tf->esp = sp;

switchuvm(curproc);

*// 같은 PID지만 주소공간이 완전히 바뀌었으므로 TLB entry 전부 폐기*

stlb\_purge\_pid(curproc->pid);

ipt\_purge\_pid(curproc->pid);

freevm(oldpgdir);

return 0;

bad:

if(pgdir)

freevm(pgdir);

if(ip){

iunlockput(ip);

end\_op();

}

return -1;

}

<main.c>

#include "types.h"

#include "defs.h"

#include "param.h"

#include "memlayout.h"

#include "mmu.h"

#include "proc.h"

#include "x86.h"

static void startothers(void);

static void mpmain(void) \_\_attribute\_\_((noreturn));

extern pde\_t \*kpgdir;

extern char end[]; *// first address after kernel loaded from ELF file*

*// Bootstrap processor starts running C code here.*

*// Allocate a real stack and switch to it, first*

*// doing some setup required for memory allocator to work.*

int

main(void)

{

kinit1(end, P2V(4\*1024\*1024)); *// phys page allocator*

kvmalloc(); *// kernel page table*

mpinit(); *// detect other processors*

lapicinit(); *// interrupt controller*

seginit(); *// segment descriptors*

picinit(); *// disable pic*

ioapicinit(); *// another interrupt controller*

consoleinit(); *// console hardware*

uartinit(); *// serial port*

pinit(); *// process table*

tvinit(); *// trap vectors*

binit(); *// buffer cache*

fileinit(); *// file table*

ideinit(); *// disk*

startothers(); *// start other processors*

kinit2(P2V(4\*1024\*1024), P2V(PHYSTOP)); *// must come after startothers()*

stlb\_init();

ipt\_init();

userinit(); *// first user process*

mpmain(); *// finish this processor's setup*

}

*// Other CPUs jump here from entryother.S.*

static void

mpenter(void)

{

switchkvm();

seginit();

lapicinit();

mpmain();

}

*// Common CPU setup code.*

static void

mpmain(void)

{

cprintf("cpu%d: starting %d\n", cpuid(), cpuid());

idtinit(); *// load idt register*

xchg(&(mycpu()->started), 1); *// tell startothers() we're up*

scheduler(); *// start running processes*

}

pde\_t entrypgdir[]; *// For entry.S*

*// Start the non-boot (AP) processors.*

static void

startothers(void)

{

extern uchar \_binary\_entryother\_start[], \_binary\_entryother\_size[];

uchar \*code;

struct cpu \*c;

char \*stack;

*// Write entry code to unused memory at 0x7000.*

*// The linker has placed the image of entryother.S in*

*// \_binary\_entryother\_start.*

code = P2V(0x7000);

memmove(code, \_binary\_entryother\_start, (uint)\_binary\_entryother\_size);

for(c = cpus; c < cpus+ncpu; c++){

if(c == mycpu()) *// We've started already.*

continue;

*// Tell entryother.S what stack to use, where to enter, and what*

*// pgdir to use. We cannot use kpgdir yet, because the AP processor*

*// is running in low memory, so we use entrypgdir for the APs too.*

stack = kalloc();

\*(void\*\*)(code-4) = stack + KSTACKSIZE;

\*(void(\*\*)(void))(code-8) = mpenter;

\*(int\*\*)(code-12) = (void \*) V2P(entrypgdir);

lapicstartap(c->apicid, V2P(code));

*// wait for cpu to finish mpmain()*

while(c->started == 0)

;

}

}

*// The boot page table used in entry.S and entryother.S.*

*// Page directories (and page tables) must start on page boundaries,*

*// hence the \_\_aligned\_\_ attribute.*

*// PTE\_PS in a page directory entry enables 4Mbyte pages.*

\_\_attribute\_\_((\_\_aligned\_\_(PGSIZE)))

pde\_t entrypgdir[NPDENTRIES] = {

*// Map VA's [0, 4MB) to PA's [0, 4MB)*

[0] = (0) | PTE\_P | PTE\_W | PTE\_PS,

*// Map VA's [KERNBASE, KERNBASE+4MB) to PA's [0, 4MB)*

[KERNBASE>>PDXSHIFT] = (0) | PTE\_P | PTE\_W | PTE\_PS,

};

*//PAGEBREAK!*

*// Blank page.*

*//PAGEBREAK!*

*// Blank page.*

*//PAGEBREAK!*

*// Blank page.*

<vtop.c>

#include "types.h"

#include "stat.h"

#include "user.h"

static uint parse\_hex(const char \**p*){

uint x=0; if(p[0]=='0'&&(p[1]=='x'||p[1]=='X')) p+=2;

for(;\*p;p++){ char c=\*p; int v;

if(c>='0'&&c<='9') v=c-'0';

else if(c>='a'&&c<='f') v=c-'a'+10;

else if(c>='A'&&c<='F') v=c-'A'+10;

else break;

x=(x<<4)|(uint)v;

} return x;

}

static void usage(void){

printf(1,"usage: vtop <hex\_va> | -s | -a N [-r M]\n");

exit();

}

int main(int *argc*, char \*\**argv*){

uint va=0; int pages=0, repeat=1; int i=1;

if(argc<2) usage();

*// 모드 선택*

if(!strcmp(argv[i],"-s")){

volatile int dummy=123; *// 최적화 방지*

va=(uint)&dummy; i++;

} else if(!strcmp(argv[i],"-a")){

if(i+1>=argc) usage();

pages=atoi(argv[i+1]); if(pages<=0) usage(); i+=2;

char \*base=sbrk(pages\*4096);

if(base==(char\*)-1){ printf(1,"vtop: sbrk failed\n"); exit(); }

for(int k=0;k<pages;k++) base[k\*4096]=(char)k; *// 실제 할당*

printf(1,"[vtop] base=0x%x pages=%d\n",(uint)base,pages);

va=(uint)base;

} else {

va=parse\_hex(argv[i]); i++;

}

*// 반복 횟수*

if(i+1<=argc && !strcmp(argv[i],"-r")){

if(i+1>=argc) usage();

repeat=atoi(argv[i+1]); if(repeat<=0) repeat=1;

}

for(int r=1;r<=repeat;r++){

uint pa=0, flags=0, hits=0, misses=0;

if(vtop((void\*)va,&pa,&flags)<0){

printf(1,"vtop: not present (VA=0x%x)\n",va);

exit();

}

tlbstat(&hits,&misses);

printf(1,"[%d] VA=0x%x -> PA=0x%x flags=0x%x TLB[hits=%d misses=%d]\n",

r, va, pa, flags, hits, misses);

}

exit();

}

<pfind.c>

**#include "types.h"**

**#include "stat.h"**

**#include "user.h"**

**static uint parse\_hex(const char \**p*){**

**uint x = 0;**

**if(p[0]=='0' && (p[1]=='x' || p[1]=='X')) p += 2;**

**while(\*p){**

**char c = \*p++;**

**int v;**

**if(c>='0' && c<='9') v = c - '0';**

**else if(c>='a' && c<='f') v = c - 'a' + 10;**

**else if(c>='A' && c<='F') v = c - 'A' + 10;**

**else break;**

**x = (x<<4) | (uint)v;**

**}**

**return x;**

**}**

**static void usage(void){**

**printf(1, "usage: pfind <hex\_pa\_page>\n");**

**exit();**

**}**

**int main(int *argc*, char \*\**argv*){**

**if(argc != 2) usage();**

**uint pa\_page = parse\_hex(argv[1]) & ~0xFFF; *// 페이지 정렬***

**struct vref buf[64];**

**int n = phys2virt(pa\_page, buf, 64);**

**if(n < 0){**

**printf(1, "pfind: syscall failed\n"); exit();**

**}**

**printf(1, "PA\_PAGE=0x%x -> %d refs\n", pa\_page, n);**

**for(int i=0;i<n;i++){**

**printf(1, " (pid=%d, va=0x%x, flags=0x%x)\n", buf[i].pid, buf[i].va, buf[i].flags);**

**}**

**exit();**

**}**

**<mmu.h>**

***// This file contains definitions for the***

***// x86 memory management unit (MMU).***

***// Eflags register***

**#define FL\_IF 0x00000200 *// Interrupt Enable***

***// Control Register flags***

**#define CR0\_PE 0x00000001 *// Protection Enable***

**#define CR0\_WP 0x00010000 *// Write Protect***

**#define CR0\_PG 0x80000000 *// Paging***

**#define CR4\_PSE 0x00000010 *// Page size extension***

***// various segment selectors.***

**#define SEG\_KCODE 1 *// kernel code***

**#define SEG\_KDATA 2 *// kernel data+stack***

**#define SEG\_UCODE 3 *// user code***

**#define SEG\_UDATA 4 *// user data+stack***

**#define SEG\_TSS 5 *// this process's task state***

***// cpu->gdt[NSEGS] holds the above segments.***

**#define NSEGS 6**

**#ifndef \_\_ASSEMBLER\_\_**

***// Segment Descriptor***

**struct segdesc {**

**uint lim\_15\_0 : 16; *// Low bits of segment limit***

**uint base\_15\_0 : 16; *// Low bits of segment base address***

**uint base\_23\_16 : 8; *// Middle bits of segment base address***

**uint type : 4; *// Segment type (see STS\_ constants)***

**uint s : 1; *// 0 = system, 1 = application***

**uint dpl : 2; *// Descriptor Privilege Level***

**uint p : 1; *// Present***

**uint lim\_19\_16 : 4; *// High bits of segment limit***

**uint avl : 1; *// Unused (available for software use)***

**uint rsv1 : 1; *// Reserved***

**uint db : 1; *// 0 = 16-bit segment, 1 = 32-bit segment***

**uint g : 1; *// Granularity: limit scaled by 4K when set***

**uint base\_31\_24 : 8; *// High bits of segment base address***

**};**

***// Normal segment***

**#define SEG(*type*, *base*, *lim*, *dpl*) (struct segdesc) \**

**{ ((lim) >> 12) & 0xffff, (uint)(base) & 0xffff, \**

**((uint)(base) >> 16) & 0xff, type, 1, dpl, 1, \**

**(uint)(lim) >> 28, 0, 0, 1, 1, (uint)(base) >> 24 }**

**#define SEG16(*type*, *base*, *lim*, *dpl*) (struct segdesc) \**

**{ (lim) & 0xffff, (uint)(base) & 0xffff, \**

**((uint)(base) >> 16) & 0xff, type, 1, dpl, 1, \**

**(uint)(lim) >> 16, 0, 0, 1, 0, (uint)(base) >> 24 }**

**#endif**

**#define DPL\_USER 0x3 *// User DPL***

***// Application segment type bits***

**#define STA\_X 0x8 *// Executable segment***

**#define STA\_W 0x2 *// Writeable (non-executable segments)***

**#define STA\_R 0x2 *// Readable (executable segments)***

***// System segment type bits***

**#define STS\_T32A 0x9 *// Available 32-bit TSS***

**#define STS\_IG32 0xE *// 32-bit Interrupt Gate***

**#define STS\_TG32 0xF *// 32-bit Trap Gate***

***// A virtual address 'la' has a three-part structure as follows:***

***//***

***// +--------10------+-------10-------+---------12----------+***

***// | Page Directory | Page Table | Offset within Page |***

***// | Index | Index | |***

***// +----------------+----------------+---------------------+***

***// \--- PDX(va) --/ \--- PTX(va) --/***

***// page directory index***

**#define PDX(*va*) (((uint)(va) >> PDXSHIFT) & 0x3FF)**

***// page table index***

**#define PTX(*va*) (((uint)(va) >> PTXSHIFT) & 0x3FF)**

***// construct virtual address from indexes and offset***

**#define PGADDR(*d*, *t*, *o*) ((uint)((d) << PDXSHIFT | (t) << PTXSHIFT | (o)))**

***// Page directory and page table constants.***

**#define NPDENTRIES 1024 *// # directory entries per page directory***

**#define NPTENTRIES 1024 *// # PTEs per page table***

**#define PGSIZE 4096 *// bytes mapped by a page***

**#define PTXSHIFT 12 *// offset of PTX in a linear address***

**#define PDXSHIFT 22 *// offset of PDX in a linear address***

**#define PGROUNDUP(*sz*) (((sz)+PGSIZE-1) & ~(PGSIZE-1))**

**#define PGROUNDDOWN(*a*) (((a)) & ~(PGSIZE-1))**

***// Page table/directory entry flags.***

**#define PTE\_P 0x001 *// Present***

**#define PTE\_W 0x002 *// Writeable***

**#define PTE\_U 0x004 *// User***

**#define PTE\_PS 0x080 *// Page Size***

***// Address in page table or page directory entry***

**#define PTE\_ADDR(*pte*) ((uint)(pte) & ~0xFFF)**

**#define PTE\_FLAGS(*pte*) ((uint)(pte) & 0xFFF)**

**#ifndef \_\_ASSEMBLER\_\_**

**typedef uint pte\_t;**

***// Task state segment format***

**struct taskstate {**

**uint link; *// Old ts selector***

**uint esp0; *// Stack pointers and segment selectors***

**ushort ss0; *// after an increase in privilege level***

**ushort padding1;**

**uint \*esp1;**

**ushort ss1;**

**ushort padding2;**

**uint \*esp2;**

**ushort ss2;**

**ushort padding3;**

**void \*cr3; *// Page directory base***

**uint \*eip; *// Saved state from last task switch***

**uint eflags;**

**uint eax; *// More saved state (registers)***

**uint ecx;**

**uint edx;**

**uint ebx;**

**uint \*esp;**

**uint \*ebp;**

**uint esi;**

**uint edi;**

**ushort es; *// Even more saved state (segment selectors)***

**ushort padding4;**

**ushort cs;**

**ushort padding5;**

**ushort ss;**

**ushort padding6;**

**ushort ds;**

**ushort padding7;**

**ushort fs;**

**ushort padding8;**

**ushort gs;**

**ushort padding9;**

**ushort ldt;**

**ushort padding10;**

**ushort t; *// Trap on task switch***

**ushort iomb; *// I/O map base address***

**};**

***// Gate descriptors for interrupts and traps***

**struct gatedesc {**

**uint off\_15\_0 : 16; *// low 16 bits of offset in segment***

**uint cs : 16; *// code segment selector***

**uint args : 5; *// # args, 0 for interrupt/trap gates***

**uint rsv1 : 3; *// reserved(should be zero I guess)***

**uint type : 4; *// type(STS\_{IG32,TG32})***

**uint s : 1; *// must be 0 (system)***

**uint dpl : 2; *// descriptor(meaning new) privilege level***

**uint p : 1; *// Present***

**uint off\_31\_16 : 16; *// high bits of offset in segment***

**};**

***// Set up a normal interrupt/trap gate descriptor.***

***// - istrap: 1 for a trap (= exception) gate, 0 for an interrupt gate.***

***// interrupt gate clears FL\_IF, trap gate leaves FL\_IF alone***

***// - sel: Code segment selector for interrupt/trap handler***

***// - off: Offset in code segment for interrupt/trap handler***

***// - dpl: Descriptor Privilege Level -***

***// the privilege level required for software to invoke***

***// this interrupt/trap gate explicitly using an int instruction.***

**#define SETGATE(*gate*, *istrap*, *sel*, *off*, *d*) \**

**{ \**

**(gate).off\_15\_0 = (uint)(off) & 0xffff; \**

**(gate).cs = (sel); \**

**(gate).args = 0; \**

**(gate).rsv1 = 0; \**

**(gate).type = (istrap) ? STS\_TG32 : STS\_IG32; \**

**(gate).s = 0; \**

**(gate).dpl = (d); \**

**(gate).p = 1; \**

**(gate).off\_31\_16 = (uint)(off) >> 16; \**

**}**

**#endif**

***// OS-private bit for COW***

**#define PTE\_COW 0x200**

**<trap.c>**

**#include "types.h"**

**#include "defs.h"**

**#include "param.h"**

**#include "memlayout.h"**

**#include "mmu.h"**

**#include "proc.h"**

**#include "x86.h"**

**#include "traps.h"**

**#include "spinlock.h"**

***// Interrupt descriptor table (shared by all CPUs).***

**struct gatedesc idt[256];**

**extern uint vectors[]; *// in vectors.S: array of 256 entry pointers***

**struct spinlock tickslock;**

**uint ticks;**

**void**

**tvinit(void)**

**{**

**int i;**

**for(i = 0; i < 256; i++)**

**SETGATE(idt[i], 0, SEG\_KCODE<<3, vectors[i], 0);**

**SETGATE(idt[T\_SYSCALL], 1, SEG\_KCODE<<3, vectors[T\_SYSCALL], DPL\_USER);**

**initlock(&tickslock, "time");**

**}**

**void**

**idtinit(void)**

**{**

**lidt(idt, sizeof(idt));**

**}**

***//PAGEBREAK: 41***

**void**

**trap(struct trapframe \**tf*)**

**{**

**if(tf->trapno == T\_SYSCALL){**

**if(myproc()->killed)**

**exit();**

**myproc()->tf = tf;**

**syscall();**

**if(myproc()->killed)**

**exit();**

**return;**

**}**

**switch(tf->trapno){**

**case T\_IRQ0 + IRQ\_TIMER:**

**if(cpuid() == 0){**

**acquire(&tickslock);**

**ticks++;**

**wakeup(&ticks);**

**release(&tickslock);**

**}**

**lapiceoi();**

**break;**

**case T\_IRQ0 + IRQ\_IDE:**

**ideintr();**

**lapiceoi();**

**break;**

**case T\_IRQ0 + IRQ\_IDE+1:**

***// Bochs generates spurious IDE1 interrupts.***

**break;**

**case T\_IRQ0 + IRQ\_KBD:**

**kbdintr();**

**lapiceoi();**

**break;**

**case T\_IRQ0 + IRQ\_COM1:**

**uartintr();**

**lapiceoi();**

**break;**

**case T\_IRQ0 + 7:**

**case T\_IRQ0 + IRQ\_SPURIOUS:**

**cprintf("cpu%d: spurious interrupt at %x:%x\n",**

**cpuid(), tf->*cs*, tf->*eip*);**

**lapiceoi();**

**break;**

**case T\_PGFLT: {**

***// fault가 난 가상주소 (CR2)***

**uint va = rcr2();**

***// 성공 경로: 조용히 처리하고 복귀***

**if ((tf->err & 0x2) && myproc() && cow\_fault(myproc()->pgdir, va) == 0)**

**return;**

***// 여기까지 왔다는 건 COW 대상이 아니거나/실패.***

***// 기존 xv6와 동일하게 프로세스 종료로 처리.***

***// 여기까지 왔으면 COW로 처리 못함 → 이때만 로그***

**#ifdef COW\_DEBUG**

**pte\_t \*pte = k\_walkpgdir(myproc()->pgdir, (void\*)PGROUNDDOWN(va));**

**if (pte)**

**cprintf("PGFLT(pid=%d) va=0x%x err=0x%x pte=0x%x flags=0x%x P=%d W=%d U=%d COW=%d\n",**

**myproc()->pid, va, tf->err, \*pte, PTE\_FLAGS(\*pte),**

**!!(\*pte&PTE\_P), !!(\*pte&PTE\_W), !!(\*pte&PTE\_U), !!(\*pte&PTE\_COW));**

**#endif**

**if (myproc())**

**myproc()->killed = 1;**

**return;**

**}**

***//PAGEBREAK: 13***

**default:**

**if(myproc() == 0 || (tf->cs&3) == 0){**

***// In kernel, it must be our mistake.***

**cprintf("unexpected trap %d from cpu %d eip %x (cr2=0x%x)\n",**

**tf->trapno, cpuid(), tf->eip, rcr2());**

**panic("trap");**

**}**

***// In user space, assume process misbehaved.***

**cprintf("pid %d %s: trap %d err %d on cpu %d "**

**"eip 0x%x addr 0x%x--kill proc\n",**

**myproc()->*pid*, myproc()->*name*, tf->*trapno*,**

**tf->*err*, cpuid(), tf->*eip*, rcr2());**

**myproc()->killed = 1;**

**}**

***// Force process exit if it has been killed and is in user space.***

***// (If it is still executing in the kernel, let it keep running***

***// until it gets to the regular system call return.)***

**if(myproc() && myproc()->killed && (tf->cs&3) == DPL\_USER)**

**exit();**

***// Force process to give up CPU on clock tick.***

***// If interrupts were on while locks held, would need to check nlock.***

**if(myproc() && myproc()->state == RUNNING &&**

**tf->trapno == T\_IRQ0+IRQ\_TIMER)**

**yield();**

***// Check if the process has been killed since we yielded***

**if(myproc() && myproc()->killed && (tf->cs&3) == DPL\_USER)**

**exit();**

**}**

**<projtest.c>**

***// projtest.c — vtop/pfind 일괄 검증***

**#include "types.h"**

**#include "stat.h"**

**#include "user.h"**

**#include "mmu.h"**

**#ifndef PGSIZE**

**#define PGSIZE 4096**

**#endif**

**static void to\_hex(uint *x*, char \**b*){**

**static const char \*D="0123456789abcdef";**

**b[0]='0'; b[1]='x';**

**for(int i=0;i<8;i++){ int s=(7-i)\*4; b[2+i]=D[(x>>s)&0xF]; }**

**b[10]=0;**

**}**

**static void run(const char \**prog*, char \*const *argv*[]){**

**int pid=fork();**

**if(pid<0){ printf(1,"fork fail\n"); return; }**

**if(pid==0){ exec(prog, argv); printf(1,"exec %s failed\n", prog); exit(); }**

**wait();**

**}**

**int**

**main(void)**

**{**

**printf(1,"\n==== projtest: vtop/pfind 통합 검증 ====\n");**

***// (A) STLB hit/miss***

**printf(1,"\n[A] STLB hit/miss 확인\n");**

**char \*a1[]={(char\*)"vtop",(char\*)"-s",(char\*)"-r",(char\*)"3",0};**

**run("vtop", a1);**

***// (B) remap/unmap → IPT/STLB***

**printf(1,"\n[B] remap/unmap → IPT/STLB 일관성\n");**

**char \*base = sbrk(PGSIZE\*2);**

**base[0]=1; base[PGSIZE]=2;**

**uint pa=0, fl=0;**

**if(vtop(base,&pa,&fl)<0){ printf(1,"vtop syscall fail\n"); exit(); }**

**char va\_hex[16], pfn\_hex[16];**

**to\_hex((uint)base, va\_hex);**

**to\_hex(pa & ~(PGSIZE-1), pfn\_hex);**

**printf(1,"[B] VA=%s PFN=%s flags=0x%x\n", va\_hex, pfn\_hex, fl);**

**char \*v1[]={(char\*)"vtop",va\_hex,0};**

**char \*p1[]={(char\*)"pfind",pfn\_hex,0};**

**run("vtop", v1);**

**run("pfind", p1);**

**sbrk(-PGSIZE\*2);**

**printf(1,"[B] 해제 후 재검증\n");**

**run("pfind", (char\*[]){(char\*)"pfind",pfn\_hex,0});**

**run("vtop", (char\*[]){(char\*)"vtop",va\_hex,0});**

***// (C) COW 공유 → 분리 → exit 정리***

**printf(1,"\n[C] COW 공유→쓰기 분리→exit 정리\n");**

**char \*p = sbrk(PGSIZE);**

**p[0]=7;**

**uint opa=0, ofl=0; vtop(p,&opa,&ofl);**

**char ofn[16]; to\_hex(opa & ~(PGSIZE-1), ofn);**

**printf(1,"[C] 원래 PFN=%s\n", ofn);**

**int cpid=fork();**

**if(cpid==0){**

***// 자식: 잠깐 대기 후 쓰기 → COW***

**sleep(20);**

**p[0]=99;**

**uint npa=0, nfl=0; vtop(p,&npa,&nfl);**

**char nfn[16]; to\_hex(npa & ~(PGSIZE-1), nfn);**

**printf(1,"[child] 쓰기 후 PFN=%s (원래와 달라야 정상)\n", nfn);**

**sleep(30);**

**exit();**

**}**

***// 부모: COW 전 2 refs***

**run("pfind", (char\*[]){(char\*)"pfind",ofn,0});**

***// 자식이 쓰기한 뒤: 1 ref***

**sleep(30);**

**printf(1,"[parent] 자식 쓰기 이후\n");**

**run("pfind", (char\*[]){(char\*)"pfind",ofn,0});**

***// 자식 종료 대기***

**wait();**

**printf(1,"[parent] 자식 exit 이후\n");**

**run("pfind", (char\*[]){(char\*)"pfind",ofn,0});**

***// 안전 처리: 부모 페이지 즉시 해제 후,***

***// 최종 pfind는 별도 프로세스가 exec로 수행***

**sbrk(-PGSIZE);**

**printf(1,"[parent] 부모 해제 이후(최종)\n");**

**run("pfind", (char\*[]){(char\*)"pfind",ofn,0});**

**printf(1,"\n==== projtest done ====\n");**

**exit();**

**}**

**<Makefile>**

**OBJS = \**

**bio.o\**

**console.o\**

**exec.o\**

**file.o\**

**fs.o\**

**ide.o\**

**ioapic.o\**

**kalloc.o\**

**kbd.o\**

**lapic.o\**

**log.o\**

**main.o\**

**mp.o\**

**picirq.o\**

**pipe.o\**

**proc.o\**

**sleeplock.o\**

**spinlock.o\**

**string.o\**

**swtch.o\**

**syscall.o\**

**sysfile.o\**

**sysproc.o\**

**trapasm.o\**

**trap.o\**

**uart.o\**

**vectors.o\**

**vm.o\**

**# Cross-compiling (e.g., on Mac OS X)**

**TOOLPREFIX = i686-linux-gnu-**

**# Using native tools (e.g., on X86 Linux)**

**#TOOLPREFIX =**

**# Try to infer the correct TOOLPREFIX if not set**

**ifndef TOOLPREFIX**

**TOOLPREFIX := $(shell if i386-jos-elf-objdump -i 2>&1 | grep '^elf32-i386$$' >/dev/null 2>&1; \**

**then echo 'i386-jos-elf-'; \**

**elif objdump -i 2>&1 | grep 'elf32-i386' >/dev/null 2>&1; \**

**then echo ''; \**

**else echo "\*\*\*" 1>&2; \**

**echo "\*\*\* Error: Couldn't find an i386-\*-elf version of GCC/binutils." 1>&2; \**

**echo "\*\*\* Is the directory with i386-jos-elf-gcc in your PATH?" 1>&2; \**

**echo "\*\*\* If your i386-\*-elf toolchain is installed with a command" 1>&2; \**

**echo "\*\*\* prefix other than 'i386-jos-elf-', set your TOOLPREFIX" 1>&2; \**

**echo "\*\*\* environment variable to that prefix and run 'make' again." 1>&2; \**

**echo "\*\*\* To turn off this error, run 'gmake TOOLPREFIX= ...'." 1>&2; \**

**echo "\*\*\*" 1>&2; exit 1; fi)**

**endif**

**# If the makefile can't find QEMU, specify its path here**

**# QEMU = qemu-system-i386**

**# Try to infer the correct QEMU**

**ifndef QEMU**

**QEMU = $(shell if which qemu > /dev/null; \**

**then echo qemu; exit; \**

**elif which qemu-system-i386 > /dev/null; \**

**then echo qemu-system-i386; exit; \**

**elif which qemu-system-x86\_64 > /dev/null; \**

**then echo qemu-system-x86\_64; exit; \**

**else \**

**qemu=/Applications/Q.app/Contents/MacOS/i386-softmmu.app/Contents/MacOS/i386-softmmu; \**

**if test -x $$qemu; then echo $$qemu; exit; fi; fi; \**

**echo "\*\*\*" 1>&2; \**

**echo "\*\*\* Error: Couldn't find a working QEMU executable." 1>&2; \**

**echo "\*\*\* Is the directory containing the qemu binary in your PATH" 1>&2; \**

**echo "\*\*\* or have you tried setting the QEMU variable in Makefile?" 1>&2; \**

**echo "\*\*\*" 1>&2; exit 1)**

**endif**

**CC = $(TOOLPREFIX)gcc**

**AS = $(TOOLPREFIX)gas**

**LD = $(TOOLPREFIX)ld**

**OBJCOPY = $(TOOLPREFIX)objcopy**

**OBJDUMP = $(TOOLPREFIX)objdump**

**CFLAGS = -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer**

**CFLAGS += $(shell $(CC) -fno-stack-protector -E -x c /dev/null >/dev/null 2>&1 && echo -fno-stack-protector)**

**ASFLAGS = -m32 -gdwarf-2 -Wa,-divide**

**# FreeBSD ld wants ``elf\_i386\_fbsd''**

**LDFLAGS += -m $(shell $(LD) -V | grep elf\_i386 2>/dev/null | head -n 1)**

**# Disable PIE when possible (for Ubuntu 16.10 toolchain)**

**ifneq ($(shell $(CC) -dumpspecs 2>/dev/null | grep -e '[^f]no-pie'),)**

**CFLAGS += -fno-pie -no-pie**

**endif**

**ifneq ($(shell $(CC) -dumpspecs 2>/dev/null | grep -e '[^f]nopie'),)**

**CFLAGS += -fno-pie -nopie**

**endif**

**xv6.img: bootblock kernel**

**dd if=/dev/zero of=xv6.img count=10000**

**dd if=bootblock of=xv6.img conv=notrunc**

**dd if=kernel of=xv6.img seek=1 conv=notrunc**

**xv6memfs.img: bootblock kernelmemfs**

**dd if=/dev/zero of=xv6memfs.img count=10000**

**dd if=bootblock of=xv6memfs.img conv=notrunc**

**dd if=kernelmemfs of=xv6memfs.img seek=1 conv=notrunc**

**bootblock: bootasm.S bootmain.c**

**$(CC) $(CFLAGS) -fno-pic -O -nostdinc -I. -c bootmain.c**

**$(CC) $(CFLAGS) -fno-pic -nostdinc -I. -c bootasm.S**

**$(LD) $(LDFLAGS) -N -e start -Ttext 0x7C00 -o bootblock.o bootasm.o bootmain.o**

**$(OBJDUMP) -S bootblock.o > bootblock.asm**

**$(OBJCOPY) -S -O binary -j .text bootblock.o bootblock**

**./sign.pl bootblock**

**entryother: entryother.S**

**$(CC) $(CFLAGS) -fno-pic -nostdinc -I. -c entryother.S**

**$(LD) $(LDFLAGS) -N -e start -Ttext 0x7000 -o bootblockother.o entryother.o**

**$(OBJCOPY) -S -O binary -j .text bootblockother.o entryother**

**$(OBJDUMP) -S bootblockother.o > entryother.asm**

**initcode: initcode.S**

**$(CC) $(CFLAGS) -nostdinc -I. -c initcode.S**

**$(LD) $(LDFLAGS) -N -e start -Ttext 0 -o initcode.out initcode.o**

**$(OBJCOPY) -S -O binary initcode.out initcode**

**$(OBJDUMP) -S initcode.o > initcode.asm**

**kernel: $(OBJS) entry.o entryother initcode kernel.ld**

**$(LD) $(LDFLAGS) -T kernel.ld -o kernel entry.o $(OBJS) -b binary initcode entryother**

**$(OBJDUMP) -S kernel > kernel.asm**

**$(OBJDUMP) -t kernel | sed '1,/SYMBOL TABLE/d; s/ .\* / /; /^$$/d' > kernel.sym**

**# kernelmemfs is a copy of kernel that maintains the**

**# disk image in memory instead of writing to a disk.**

**# This is not so useful for testing persistent storage or**

**# exploring disk buffering implementations, but it is**

**# great for testing the kernel on real hardware without**

**# needing a scratch disk.**

**MEMFSOBJS = $(filter-out ide.o,$(OBJS)) memide.o**

**kernelmemfs: $(MEMFSOBJS) entry.o entryother initcode kernel.ld fs.img**

**$(LD) $(LDFLAGS) -T kernel.ld -o kernelmemfs entry.o $(MEMFSOBJS) -b binary initcode entryother fs.img**

**$(OBJDUMP) -S kernelmemfs > kernelmemfs.asm**

**$(OBJDUMP) -t kernelmemfs | sed '1,/SYMBOL TABLE/d; s/ .\* / /; /^$$/d' > kernelmemfs.sym**

**tags: $(OBJS) entryother.S \_init**

**etags \*.S \*.c**

**vectors.S: vectors.pl**

**./vectors.pl > vectors.S**

**ULIB = ulib.o usys.o printf.o umalloc.o**

**\_%: %.o $(ULIB)**

**$(LD) $(LDFLAGS) -N -e main -Ttext 0 -o $@ $^**

**$(OBJDUMP) -S $@ > $\*.asm**

**$(OBJDUMP) -t $@ | sed '1,/SYMBOL TABLE/d; s/ .\* / /; /^$$/d' > $\*.sym**

**\_forktest: forktest.o $(ULIB)**

**# forktest has less library code linked in - needs to be small**

**# in order to be able to max out the proc table.**

**$(LD) $(LDFLAGS) -N -e main -Ttext 0 -o \_forktest forktest.o ulib.o usys.o**

**$(OBJDUMP) -S \_forktest > forktest.asm**

**mkfs: mkfs.c fs.h**

**gcc -Werror -Wall -o mkfs mkfs.c**

**# Prevent deletion of intermediate files, e.g. cat.o, after first build, so**

**# that disk image changes after first build are persistent until clean. More**

**# details:**

**# http://www.gnu.org/software/make/manual/html\_node/Chained-Rules.html**

**.PRECIOUS: %.o**

**UPROGS=\**

**\_cat\**

**\_echo\**

**\_forktest\**

**\_grep\**

**\_init\**

**\_kill\**

**\_ln\**

**\_ls\**

**\_mkdir\**

**\_rm\**

**\_sh\**

**\_stressfs\**

**\_usertests\**

**\_wc\**

**\_zombie\**

**\_memdump\**

**\_memstress\**

**\_memtest\**

**\_vtop\**

**\_pfind\**

**\_projtest\**

**fs.img: mkfs README $(UPROGS)**

**./mkfs fs.img README $(UPROGS)**

**-include \*.d**

**clean:**

**rm -f \*.tex \*.dvi \*.idx \*.aux \*.log \*.ind \*.ilg \**

**\*.o \*.d \*.asm \*.sym vectors.S bootblock entryother \**

**initcode initcode.out kernel xv6.img fs.img kernelmemfs \**

**xv6memfs.img mkfs .gdbinit \**

**$(UPROGS)**

**# make a printout**

**FILES = $(shell grep -v '^\#' runoff.list)**

**PRINT = runoff.list runoff.spec README toc.hdr toc.ftr $(FILES)**

**xv6.pdf: $(PRINT)**

**./runoff**

**ls -l xv6.pdf**

**print: xv6.pdf**

**# run in emulators**

**bochs : fs.img xv6.img**

**if [ ! -e .bochsrc ]; then ln -s dot-bochsrc .bochsrc; fi**

**bochs -q**

**# try to generate a unique GDB port**

**GDBPORT = $(shell expr `id -u` % 5000 + 25000)**

**# QEMU's gdb stub command line changed in 0.11**

**QEMUGDB = $(shell if $(QEMU) -help | grep -q '^-gdb'; \**

**then echo "-gdb tcp::$(GDBPORT)"; \**

**else echo "-s -p $(GDBPORT)"; fi)**

**ifndef CPUS**

**CPUS := 2**

**endif**

**QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp $(CPUS) -m 512 $(QEMUEXTRA)**

**qemu: fs.img xv6.img**

**$(QEMU) -serial mon:stdio $(QEMUOPTS)**

**qemu-memfs: xv6memfs.img**

**$(QEMU) -drive file=xv6memfs.img,index=0,media=disk,format=raw -smp $(CPUS) -m 256**

**qemu-nox: fs.img xv6.img**

**$(QEMU) -nographic $(QEMUOPTS)**

**.gdbinit: .gdbinit.tmpl**

**sed "s/localhost:1234/localhost:$(GDBPORT)/" < $^ > $@**

**qemu-gdb: fs.img xv6.img .gdbinit**

**@echo "\*\*\* Now run 'gdb'." 1>&2**

**$(QEMU) -serial mon:stdio $(QEMUOPTS) -S $(QEMUGDB)**

**qemu-nox-gdb: fs.img xv6.img .gdbinit**

**@echo "\*\*\* Now run 'gdb'." 1>&2**

**$(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)**

**# CUT HERE**

**# prepare dist for students**

**# after running make dist, probably want to**

**# rename it to rev0 or rev1 or so on and then**

**# check in that version.**

**EXTRA=\**

**mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\**

**ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\**

**printf.c umalloc.c\**

**README dot-bochsrc \*.pl toc.\* runoff runoff1 runoff.list\**

**.gdbinit.tmpl gdbutil\**

**dist:**

**rm -rf dist**

**mkdir dist**

**for i in $(FILES); \**

**do \**

**grep -v PAGEBREAK $$i >dist/$$i; \**

**done**

**sed '/CUT HERE/,$$d' Makefile >dist/Makefile**

**echo >dist/runoff.spec**

**cp $(EXTRA) dist**

**dist-test:**

**rm -rf dist**

**make dist**

**rm -rf dist-test**

**mkdir dist-test**

**cp dist/\* dist-test**

**cd dist-test; $(MAKE) print**

**cd dist-test; $(MAKE) bochs || true**

**cd dist-test; $(MAKE) qemu**

**# update this rule (change rev#) when it is time to**

**# make a new revision.**

**tar:**

**rm -rf /tmp/xv6**

**mkdir -p /tmp/xv6**

**cp dist/\* dist/.gdbinit.tmpl /tmp/xv6**

**(cd /tmp; tar cf - xv6) | gzip >xv6-rev10.tar.gz # the next one will be 10 (9/17)**

**.PHONY: dist-test dist**