/include/linux/pipe_fs_i.h

https://elixir.bootlin.com/linux/v5.8/source/include/linux/pipe_fs_i.h

1. 매크로 및 변수

가. _LINUX_PIPE_FS_I_H

#ifndef _LINUX_PIPE_FS_I_H 과 함께
헤더의 중복 로드를 방지하기 위한 테크닉으로 사용된다.

나. PIPE_DEF_BUFFERS

#define PIPE_DEF_BUFFERS 16
파이프의 기본 버퍼 크기를 16으로 정의

다. PIPE_BUF_FLAGs

파이프 버퍼의 메모리 관리, 데이터 전송방식, read() / write() 함수의 동작 제어를 위한 플래그 값 정의

1) #define PIPE_BUF_FLAG_LRU 0x01 /* page is on the LRU */
해당 페이지가 LRU(Least Recently Used) 캐시에서 관리되고 있을 경우,
파이프 버퍼의 플래그 값을 0x01로 정의

2) #define PIPE_BUF_FLAG_ATOMIC 0x02 /* was atomically mapped */
해당 페이지가 atomically mapping(단일 CPU 명령) 되었을 때,
파이프 버퍼의 플래그 값을 0x02로 정의

3) #define PIPE_BUF_FLAG_GIFT 0x04 /* page is a gift */
해당 페이지가 gift page(다른 프로세스가 파이프 버퍼에 제공한 페이지) 일 때,
파이프 버퍼의 플래그 값을 0x04로 정의

4) #define PIPE_BUF_FLAG_PACKET 0x08 /* read() as a packet */
해당 페이지를 패킷으로 읽어야 할 때,
파이프 버퍼의 플래그 값을 0x08로 정의

5) #define PIPE_BUF_FLAG_CAN_MERGE 0x10 /* can merge buffers */
파이프 버퍼가 다른 버퍼와 병합이 가능할 때,
파이프 버퍼의 플래그 값을 0x10으로 정의
이 플래그 값이 설정된 경우 커널은 메모리 절약을 위해 여러 개의 작은 버퍼를 더 큰 단일 버퍼로 병합이 가능하다.

6) #define PIPE_BUF_FLAG_WHOLE 0x20 /* read() must return entire buffer or error */
read() 함수가 전체 버퍼를 읽어야 할 때,
파이프 버퍼의 플래그 값을 0x20으로 정의
이 플래그 값이 설정된 경우 read() 함수는 해당 버퍼의 전체 내용을 읽어야하고, 일부만 읽고 함수를 종료하는 것이 불가능하다.

7) #define PIPE_BUF_FLAG_LOSS 0x40 /* Message loss happened after this buffer */
#ifdef CONFIG_WATCH_QUEUE 와 함께 사용되어
프로세스가 파일 시스템 이벤트를 모니터링 하는 것이 가능하다면,
메모리 로스가 발생 했을 때 파이프 버퍼의 플래그 값을 0x40으로 정의

라. PIPE_SIZE

#define PIPE_SIZE PAGE_SIZE
실제 메모리 할당의 길이
PIPE_BUF는 원자성을 보장하는 것이니 다름에 유의

마. 외부 변수

1) extern unsigned int pipe_max_size;
   /fs/pipe.c에서 정의
   루트가 아닌 사용자가 파이프를 확장할 수 있는 최대 크기
   /proc/sys/fs/fs/pipe-max-size에서 루트로 설정 가능
2) extern unsigned long pipe_user_pages_hard;
   /fs/pipe.c에서 정의
   사용자 당 할당 가능한 최대 페이지 수
   소프트는 기본 값과 일치함
3) extern unsigned long pipe_user_pages_soft;
   /fs/pipe.c에서 정의
   사용자 당 할당 가능한 최대 페이지 수
   하드는 기본적으로 설정되지 않음

2. 구조체

가. struct pipe_buffer

파이프 버퍼를 나타내는 구조체

다음의 멤버변수를 포함함

1) struct page *page;
   데이터가 저장된 페이지를 가리키는 포인터
2) unsigned int offset, len;
   offset- 페이지 내에서 데이터의 시작을 나타내는 변수
   len- 페이지 내에서 데이터의 길이를 나타내는 변수
3) const struct pipe_buf_operations *ops;
   이 버퍼와 연결된 작업을 정의하는 struct pipe_buf_operation 구조체에 대한 포인터
4) unsigned int flags;
   파이프 버퍼의 플래그를 나타내는 값
5) unsigned long private;
   연관된 작업(ops)에 의해 사용되는 개인 데이터를 저장하는 변수

나. struct pipe_inode_info

파이프의 i-node 정보를 나타내는 구조체

파이프에 대한 정보 저장과 동작제어를 위한 함수를 제공

1) struct mutex mutex;
   파이프에 대한 동시성 접근 제어
2) wait_queue_head_t rd_wait, wr_wait;
   파이프에 데이터가 없을 때, reader가 기다리기 위한 wait queue
   파이프에 데이터가 가득 찼을 때, writer가 기다리기 위한 wait queue
3) unsigned int head;
   파이프 버퍼의 head 위치
   다음에 읽을 데이터의 위치를 가리킴
4) unsigned int tail;
   파이프 버퍼의 tail 위치
   다음에 데이터를 쓸 위치를 가리킴
5) unsigned int max_usage;
   파이프 버퍼의 최대 사용량
   파이프에 저장할 수 있는 데이터의 최대 크기를 결정
6) unsigned int ring_size;
   파이프 버퍼의 크기
   파이프에 사용할 수 있는 버퍼 개수를 결정함
7) bool note_loss;
   (CONFIG_WATCH_QUEUE 구성이 활성화 되어 있을 때 포함됨)
   다음 read() 호출 시 데이터 손실 메시지를 삽입해야 함을 나타내는 플래그
8) unsigned int nr_accounted;
   현재 사용자 프로세스가 사용하는 파이프 버퍼의 수
9) unsigned int readers;
   파이프에서 데이터를 읽는 프로세스의 수
10) unsigned int writers;
    파이프에서 데이터를 쓰는 프로세스의 수
11) unsigned int files;
    파이프를 참조하는 파일의 개수

12) unsigned int r_counter;
reader 카운터
13) unsigned int w_counter;
writer 카운터
14) unsigned int poll_usage;
파이프가 epoll에 사용되고 있는지 여부를 나타냄
epoll: 이베트 기반 I/O 시스템, 풀링 대기 시간 최소화를 위해 이벤트 발생 때까지 프로세스를 대기
epoll에 사용될 경우 즉각적인 wakeup에 대응해야 되기에 별도의 처리가 수행되어야 함,
때문에 epoll에 사용되는지 여부를 구분하여 리소스 사용량을 최적화 하고자 함
15) struct page *tmp_page;
임시적으로 사용되는 페이지
16) struct fasync_struct *fasync_readers;
reader 측의 fasync 구조체(비동기 I/O 이벤트의 감지에 사용됨)
17) struct fasync_struct *fasync_writers;
writer 측의 fasync 구조체
18) struct pipe_buffer *bufs;
파이프 버퍼의 배열
19) struct user_struct *user;
파이프를 생성한 사용자 프로세스
20) struct watch_queue *watch_queue;
(CONFIG_WATCH_QUEUE 구성 옵션이 활성화 되어 있을 때 포함됨)
파이프가 watch queue로 사용되는 경우, watch queue 관련 정보를 저장하는 포인터

다. struct pipe_buf_operations

파이프 버퍼에 대한 작업을 정의하는 구조체
confirm / release / try_steal / get 의 4개의 멤버 함수를 포함함
이 중 confirm 함수는 try_steal 함수의 실행 전에 반드시 실행되어야 함
대상 버퍼가 데이터가 유효하고 사용 가능한지를 판단해야 하기 때문

1) int (*confirm)(struct pipe_inode_info *, struct pipe_buffer *);
파이프 버퍼의 데이터가 유효하고 사용 가능한지 확인하는 함수
파일 시스템에 속하는 페이지를 포함하는 파이프 버퍼의 경우, I/O 완료를 기다려야 할 수도 있음
성공할 경우 0 반환, 실패 시 음수로 에러 값 반환
2) void (*release)(struct pipe_inode_info *, struct pipe_buffer *);
파이프 버퍼의 내용이 완전히 소비되었을 때 호출되는 함수
버퍼의 메모리 자원을 해제하는 작업을 수행
3) bool (*try_steal)(struct pipe_inode_info *, struct pipe_buffer *);
파이프 버퍼의 소유권을 획득하려고 시도하는 함수
성공 시 버퍼의 내용을 잠그고 호출자가 완전히 소유하게 함
이후 페이지를 다른 매핑으로 전송 가능
성공 시 true 반환, 실패 시 false 반환
4) bool (*get)(struct pipe_inode_info *, struct pipe_buffer *);
파이프 버퍼에 대한 참조를 얻는 함수
버퍼의 사용 횟수를 증가시킴
성공하면 true 반환, 실패 시 false 반환

라. 외부 구조체

1) extern const struct pipe_buf_operations nosteal_pipe_buf_ops;
/fs/splice.c에서 정의
소켓 등을 위한 pipe_buffer_operations을 정의

3. inline 함수

호출되는 함수가 아닌 매크로처럼 사용된 부분에 스크립트가 붙여 넣어지는 함수들
함수 호출 스택 등을 아낄 수 있는 구조임

가. bool pipe_empty(unsigned int head, unsigned int tail)
파이프가 비어 있다면 true를 반환하는 함수

나. unsigned int pipe_occupancy(unsigned int head, unsigned int tail)
파이프가 사용 중인 slot의 개수를 반환하는 함수

다. bool pipe_full(unsigned int head, unsigned int tail, unsigned int limit)
파이프가 가득 차 있다면 true를 반환하는 함수

라. unsigned int pipe_space_for_user(unsigned int head, unsigned int tail, struct pipe_inode_info *pipe)
파이프의 slot 중 userspace로 사용가능한 개수를 반환하는 함수

마. __must_check bool pipe_buf_get(struct pipe_inode_info *pipe, struct pipe_buffer *buf)
pipe_buf_operations get 함수 호출
파이프 버퍼에 대한 참조를 반환하는 함수

바. void pipe_buf_release(struct pipe_inode_info *pipe, struct pipe_buffer *buf)
pipe_buf_operations release 함수 호출
파이프 버퍼의 메모리 자원을 해제하는 작업 수행

사. int pipe_buf_confirm(struct pipe_inode_info *pipe, struct pipe_buffer *buf)
pipe_buf_operations confirm 함수 호출
파이프 버퍼의 데이터가 유효하고 사용 가능한지 확인
성공 시 0 반환

아. bool pipe_buf_try_steal(struct pipe_inode_info *pipe, struct pipe_buffer *buf)
pipe_buf_operations try_steal 함수 호출
파이프 버퍼의 소유권을 획득하려고 시도하는 함수
성공 시 true 반환

4. 함수 원형

static, inline 이 아닌 함수들을 <linux/pipe_fs_i.h>를 include하였을 때, 사용가능하도록 원형들을 정의
실제 구현은 /fs/pipe.c에서 구현

가. lock

pipe의 락 기능 함수들

1) void pipe_lock(struct pipe_inode_info *);
2) void pipe_unlock(struct pipe_inode_info *);
3) void pipe_double_lock(struct pipe_inode_info *, struct pipe_inode_info *);

나. wait 및 정보관리

1) void pipe_wait_readable(struct pipe_inode_info *);
   파이프가 readable할 때까지 대기하는 함수
2) void pipe_wait_writable(struct pipe_inode_info *);
   파이프가 writable할 때까지 대기하는 함수
3) struct pipe_inode_info *alloc_pipe_info(void);
   새로운 pipe_inode_info 구조체를 할당 -> 파이프에 대한 정보 저장과 동작 관리용
4) void free_pipe_info(struct pipe_inode_info *);
   pipe_inode_info 구조체를 해제함

다. 파이프 버퍼 동작 함수

1) bool generic_pipe_buf_get(struct pipe_inode_info *, struct pipe_buffer *);
   파이프에서 사용 가능한 버퍼를 가져옴
   데이터를 쓰기 전에 호출됨
2) bool generic_pipe_buf_try_steal(struct pipe_inode_info *, struct pipe_buffer *);
   파이프 버퍼를 독점하기 위한 함수
   다른 프로세스의 사용 전에 파이프 버퍼를 가져오는데 사용
3) void generic_pipe_buf_release(struct pipe_inode_info *, struct pipe_buffer *);
   파이프 버퍼를 해제
   파이프에 데이터를 쓰기 완료 후, 해당 버퍼를 반환하는데 사용

라. 파이프 크기 조정 및 파일 관리 함수

파이프 파일의 F_SETPIPE_SZ 및 F_GET_PIPE_SZ 명령을 처리

1) int pipe_resize_ring(struct pipe_inode_info *pipe, unsigned int nr_slots);
   파이프의 링 버퍼 크기를 조정
2) long pipe_fcntl(struct file *, unsigned int, unsigned long arg);
   파이프 파일의 F_SETPIPE_SZ 및 F_GET_PIPE_SZ 명령을 처리
3) struct pipe_inode_info *get_pipe_info(struct file *file, bool for_splice);
   struct file *file 객체에서 pie_inode_info 구조체를 가져옴
4) int create_pipe_files(struct file **, int);
   두 개의 파이프 파일을 생성함
5) unsigned int round_pipe_size(unsigned long size);
   파이프 크기를 올림처리 함

마. 기타 함수

CONFIG_WATCH_QUEUE 가 활성화 되어 있을 때, 정의됨

1) unsigned long account_pipe_buffers(struct user_struct *user, unsigned long old, unsigned long new);

파이프 버퍼 사용량을 계산

2) bool too_many_pipe_buffers_soft(unsigned long user_bufs);
(soft) 사용자 프로세스가 너무 많은 파이프버퍼를 사용하고 있는지 확인

3) bool too_many_pipe_buffers_hard(unsigned long user_bufs);
(hard) 사용자 프로세스가 너무 많은 파이프 버퍼를 사용하고 있즞니 확인

4) bool pipe_is_unprivileged_user(void);
현재 프로세스가 권한이 없는 사용자인지 확인

/fs/pipe.c