Flow Kernel APIs— WQ-g15

get active jobs ioctl [./async]

1. Acquire active_jobs list lock

2. Traverse Active Jobs List
2a. Check job_status field of wrapper struct, if complete, move on
2b. Call WQ job status(pending and busy) Api for every work_struct
3. Keep updating an list_all_jobs struct => Add relevant fields for those Jobs which are not completed

4. Release lock
5. Return list_all_jobs struct

active_jobs List

1a. Acquire active_jobs list lock
1b, Acquire completed_jobs_ll list lock

2. Traverse active_jobs list
2a. If work_struct_wrapper field job_status is set to COMPLETE, or if !work_pending && !work_busy
2aa. Create a path string {jobid.out} and write the result of the job to this file
2aaa. In case of DELETE, STAT, RENAME, also copy args from wrapper to write the filenames with corresponding status, else write from op_status.
2ab. Allocate a completed_jobs struct
2ac. Transfer relevant fields from work_struct_wrapper to completed_jobs
2ada. If op=DELETE or op=STAT or op=RENAME: Allocate var_arg with argc size argv and status - int array of argc size
2adb. Copy status and arg array(special care with RENAME)
2ae. Safe delete work_struct_wrapper node from active_jobs list and dealloc node
2af. Add completed_jobs node to completed_jobs_ll tail
2ag. If completed_jobs_ll_size++ == MAX_SIZE_COMPLETED_JOBS_LL, safe remove(&dealloc) head node from completed_jobs_ll list, completed_jobs_ll_size--
2b. Else, continue

3. Release completed_jobs_ll list lock
4. Release active_jobs list lock

job list swapper function

completed_jobs List

Remove Job with jobid [./async remove {jobid}]

1. Acquire active_jobs List Lock
2. Safe Traverse Active Jobs List
3. Check iteratively for jobID in wrapper struct
3b. If found— Check job_status: call wq pending api, if success: call sync cancel job api of WQ: store api's return value

4a. Dealloc wrapper struct, after deallocing var args feld and safe emove from Active Jobs List
4b. Release active_jobs list lock

5. If JobID not found anywhere, return GENERIC error
6. Return to user space

0. Call job list swapper function
1. Acquire complete jobs list lock
2. Traverse complete jobs list, populate list_all_completed_jobs struct, max 20 entries
3. Populate count with the number of entries traversed
4. Release lock
5. Return to user space

get_completed ioctl [./async list_completed]

workq.Boost Priority [./async -p {jobID}]

1. Acquire active_jobs List Lock
2a. Safe Traverse Active Jobs List
2b. Check iteratively for jobID in wrapper struct
2c. If found— Check job_status: Call wq pending api, if success Check priority field of wrapper struct, if high DON'T DO ANYTHING. If low, call wq cancel sync api to remove job from low priority wq.
2d. If api's response successful : Add work_struct to High WQ (usinq queue work api)
4b. If api response success, update priority field of wrapper struct to high
4c. Release active_jobs list lock

5. If JobID not found anywhere, return GENERIC error
6. Return to user space

Low Priority WQ

High priority WQ

Poll Status of a job [./async -s {jobID}]

0. Alloc a poll_status struct
1. Call job list swapper function
2. Try to populate poll_Status from completed list
2a. Acquire complete_jobs_ll list lock
2b. Traverse completed list, If job found in this list, populate poll_status struct and return success
2c. If not found in the list, return GENERIC error
2d. Release completed list lock
3. If step 2 was failure, populate poll_status struct from active list.
3a. Acquire active list lock
3b. Safe traverse active list looking for jobID
3c. If found populate poll_status
3d. Release active list lock

4. Return to poll_status struct to user space, if failure return error code

Generic operation ioctl [./async]

0. Alloc a wrapper_struct
1. Initialise basic fields, like job_id, user_id, job_status, operation, priority, time of queueing
1. Alloc correctly casted wrapper void* corresponding to operation and copy arguments form user to it
2. Intiate wrapper list and add check if active list has max pending/running nodes already.
3. If not add this node to the tail of active jobs linked list
4. Initialise wrapper's work struct "async_job" with corresponding operation worker
5. Add this async_job work_struct to workqueue
6. Return job_id to user space on success

GLOBAL VARIABLES
completed_jobs_ll_size - unsigned int
completed_list_mutex - mutex
active_list_mutex - mutex

list api return structs

list_all_jobs
count - int
jobs[count]

jobs
user_id - int
job_id - int
job_status - int
op - int
priority - int
time_on_queue - time_t

list_all_completed_jobs
count - int (Max: 20)
jobIDs - int[count]

poll_status
job_id(for input) - int
job_status - int
user_id - int
count - int
filenames - char*[]
status (success/error) - int[argc]
op - int
op_status - int

Meta data Linked list structs

work_struct_wrapper
list - list_head
async_job - work_struct
op - int
user_id - int
job_id - int
priority - int
void* args (op dependent)
job_status - int
status(success/error per file) - int[argc]
op_status (if op!=STAT||DELETE||RENAME) - int
#EC: TimeMetadata

completed_jobs_node
list - list_head
user_id - int
job_id - int
var_arg (filenames)
status (success/error per file) - int[argc]
op_status (if op!=STAT||DELETE||RENAME) - int
op - int

void* arg

Utility stuct

var_arg
argc - int
argv - char*[argc]

Input structs

enc_dec_arg Struct
input_file_path - char *
output_file_path - char *
key - char *
keylen - int
flag - int

rename_files Struct
vat_args - var_arg (for input files)
dest_file_paths - char*[argc]

delete_files Struct = var_arg

stat_files Struct = var_arg

concat_files
argc - int
input_file_paths - char*[argc]
output_file_path - char *

hash_file Struct
input_file_path - char *
output_file_path - char *
file_hash - char* (to return)

compress_file
input_file_path - char *
output_file_path - char *
flag - int