Duration: 2 hour 30 min

Examiner: A. Goel

Please fill your student number and name below and then read the instructions below carefully.

Student Number: |__|__|__|__|__|__|__|__|__|__|

First Name: |__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Last Name: |__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Instructions

Examination Aids: No examination aids are allowed.

Do not turn this page until you have received the signal to start.

Do not remove any sheets from this test book. Answer all questions in the space provided. No additional sheets are permitted. Use the last blank page as scratch space. This page will not be marked.

This exam consists of 9 questions on 15 pages (including this page). The value of each part of each question is indicated. The total value of all questions is 100.

For the written answers, explain your reasoning clearly. Be as brief and specific as possible. Clear, concise answers will be given higher marks than vague, wordy answers. Marks will be deducted for incorrect statements in an answer. Please write legibly!

Work independently.

MARKING GUIDE

Q1:	_______	(12)
Q2:	_______	(12)
Q3:	_______	(10)
Q4:	_______	(8)
Q5:	_______	(10)
Q6:	_______	(14)
Q7:	_______	(6)
Q8:	_______	(18)
Q9:	_______	(10)
TOTAL:	_______	(100)

Question 1. True / False [12 MARKS]

TRUE [✓ FALSE] A process, running in user mode, cannot access a memory-mapped device even when the device is mapped to the process's memory.

TRUE [✓ FALSE] Device hardware can invoke a system call by executing the trap instruction.

[✓ TRUE] FALSE In Unix, when a kernel thread opens a file for reading, other threads of the process can also read data from the open file.

TRUE [✓ FALSE] A multi-threaded web server program will not run any faster than a single-threaded web server program on a single core machine.

[✓ TRUE] FALSE The `setcontext` call restores the interrupt (signal) state to the state when the corresponding `getcontext` call was made.

TRUE [✓ FALSE] When Process A issues a `kill` system call on a different Process B, then B may run code located in A's address space.

TRUE [✓ FALSE] The page fault frequency algorithm allocates enough memory to each process to meet its working set demands.

[✓ TRUE] FALSE When a program generates a "segmentation fault", it has accessed an invalid virtual address.

TRUE [✓ FALSE] Demand paging cannot be implemented if the size of the swap area is smaller than the total size of physical memory.

[✓ TRUE] FALSE With undo recovery, once a block update is done or committed, then no update need to be performed on it during recovery.

[✓ TRUE] FALSE The Berkeley fast file system improves file system performance by co-locating the directory entry, inode and the blocks of a file on the same cylinder.

[✓ TRUE] FALSE The purpose of the shuffle operation in MapReduce is to group intermediate data by their key values.

Question 2. Multiple Choice [12 MARKS]

For each of the questions below, circle all the correct answers. You start with 1.5 mark for each question, and lose 0.5 mark for each incorrect answer (non-answer or wrong answer), for a minimum of 0 marks.

A) Which of the following scheduling policies do not have issues with starvation?

[✓ 1.] Least recently scheduled

2. Multi-level queue scheduling

3. Last in, first out

4. Static priority scheduling

5. Shortest job first

[✓ 6.] Dynamic priority

B) Which of these operations will always cause a mode switch?

[✓ 1.] Program invokes `getpid()` system call

2. Jump to an address in the stack area

3. Program calls `malloc()`

[✓ 4.] OS invokes signal handler function in user program

[✓ 5.] A page fault occurs

6. Kernel thread scheduler invokes context switch

C) Which of these operations, running in the kernel, may cause the current thread to block?

1. Acquire spin_lock

2. Wakeup thread

[✓ 3.] Acquire mutex_lock

4. Disable interrupt

[✓ 5.] Allocate memory for user process

6. Send message to device

D) Which of the following are shared between two threads of a process, currently running on two different cores?

[✓ 1.] Stack region

2. Registers

[✓ 3.] Page table

4. TLB

[✓ 5.] Heap region

[✓ 6.] Code region

E) Which of the following OS features can improve the performance of a CPU-intensive, multi-threaded process?

[✓ 1.] Processor affinity

2. Swapping

3. File buffer cache

No deduction of marks if 3 was marked, because it may help as well, depending on whether the process is performing any file I/O.

[✓ 4.] Page prefetching

5. Preemptive scheduling

[✓ 6.] Load balancing

F) Which of the following may cause changes to the page table of a process (assume page eviction is disabled)?

1. Context switch

[✓ 2.] Call a function

3. Return from function call to its caller function

[✓ 4.] Copy-on-write fault

[✓ 5.] Invoke `malloc()`

6. Invoke `write()` system call with an initialized buffer

G) Which of the following are benefits of demand paging?

1. Programs can run directly from disk

[✓ 2.] A program whose size is larger than physical memory can be run

3. Programs run faster than on a system without demand paging

[✓ 4.] Programs can be written without knowledge of the physical memory size

5. Programs can be written without knowledge of the virtual address space size

6. Programs are guaranteed that memory allocation will always succeed

H) Which of the following are benefits of memory-mapped files compared to reading and writing files?

[✓ 1.] Fewer memory copies are required for accessing a file

2. Fewer disk accesses are required for accessing a file

[✓ 3.] Spin locks between processes can be implemented entirely at the user level

4. Mutex locks between processes can be implemented entirely at the user level

5. Programs have greater control over when file data is flushed to disk

6. Files whose total size is larger than the physical memory size can be accessed

Question 3. Reader-Writer Locks [10 MARKS]

Part (a) [8 MARKS] A reader-writer lock is similar to a mutex lock, but it allows multiple concurrent readers to access shared data. When any reader accesses the shared data, a writer cannot access the data. When a writer accesses the shared data, then no reader can access the data. Implement a reader-writer lock using a mutex lock. We have provided the entire code for the `rwlock` structure and for creating a `rwlock`. You need to implement the lock and unlock routines for the readers and the writers. Do not worry about starvation in your implementation. However, your solution should not have a deadlock, and it should not use spinning.

```
struct rwlock {
    struct lock *read, *write;
    int nr_read;
};
```

```
struct rwlock *rwlock_create()
{
    unsigned sz = sizeof(struct rwlock);
    struct rwlock *ret = malloc(sz);
    ret->read = lock_create();
    ret->write = lock_create();
    ret->nr_read = 0;
    return ret;
}
```

```
void r_lock(struct rwlock *rwlock)
{
    lock_acquire(rwlock->read);
    rwlock->nr_read++;
    if (rwlock->nr_read == 1) {
        lock_acquire(rwlock->write);
    }
    lock_release(rwlock->read);


}
```

```
void r_unlock(struct rwlock *rwlock)
{
    lock_acquire(rwlock->read);
    rwlock->nr_read--;
    if (rwlock->nr_read == 0) {
        lock_release(rwlock->write);
    }
    lock_release(rwlock->read);


}
```

```
void w_lock(struct rwlock *rwlock)
{
   lock_acquire(rwlock->write);




}
```

```
     w_unlock
void ~~w_lock~~(struct rwlock *rwlock)
{
    lock_release(rwlock->write);




}
```

Part (b) [2 MARKS] In Lab 5, suppose you had used a single mutex lock to protect your file cache in your caching webserver, and you had implemented the LRU eviction policy. Would there be benefits to replacing the mutex lock with a reader-writer lock? Clearly explain your answer.

The cache needs to be modified at each request to implement LRU eviction. Replacing a single mutex lock with a reader-writer lock would not provide any benefits because each request would require using a writer_lock (w_lock).

Question 4. Thread Join [8 MARKS]

Suppose you had to add a thread join functionality to Lab 3, similar to the waitpid() system call. Consider the following specification:

```
Tid thread_join(Tid tid)
```

The `thread_join()` function waits for the thread specified by `tid` to exit. Multiple threads are allowed to issue `thread_join()` on a given `tid`. The function returns `tid` on success. The function immediately returns with a `THREAD_FAILED` status, if `tid` has terminated, or if a deadlock can occur. Consider the following implementation:

```
struct thread {
    Tid tid;                    // the tid of this thread
    struct wait_queue *queue; // list of threads waiting for this thread
    struct thread *wait;      // the thread you are waiting for
};
struct thread *curthread;     // global pointer to current thread

// returns a thread specified by tid, returns NULL if tid doesn’t exist
struct thread *thread_get(Tid tid);

// determines if a deadlock is possible
int can_deadlock(struct thread *thread) {
    return (thread->wait == curthread);
}

Tid thread_join(Tid tid) {
    int enabled = interrupts_off();
    struct thread *thread = thread_get(tid);
    // thread doesn’t exist or deadlock possible
    if (!thread || can_deadlock(thread)) {
        tid = THREAD_FAILED;
        goto done;
    }
    curthread->wait = thread;     // wait for thread
    thread_sleep(thread->queue); // sleep on the thread’s queue
    curthread->wait = NULL;       // thread has exited
done:
    interrupts_set(enabled);
    return tid;
}

Tid thread_exit(void) {
    ...
    thread_wakeup(curthread->queue, THREAD_ALL);
    ...
}
```

Part (a) [2 MARKS] The current implementation attempts to address a possible deadlock situation using the `can_deadlock` function. Describe a deadlock scenario that this function handles.

When Thread A performs a join on Thread B, Thread B has already performed a join on Thread A.

Part (b) [2 MARKS] Describe a deadlock scenario that the current `can_deadlock` function cannot detect.

1. If Thread A performs a join on itself.
2. When Thread A performs a join on Thread B, Thread B has performed a join on Thread C and Thread C has performed a join on Thread A, i.e., a cycle with greater than 2 threads is formed.

Part (c) [4 MARKS] Replace the `can_deadlock` function with your own implementation so that it can detect any situation where deadlock can arise.

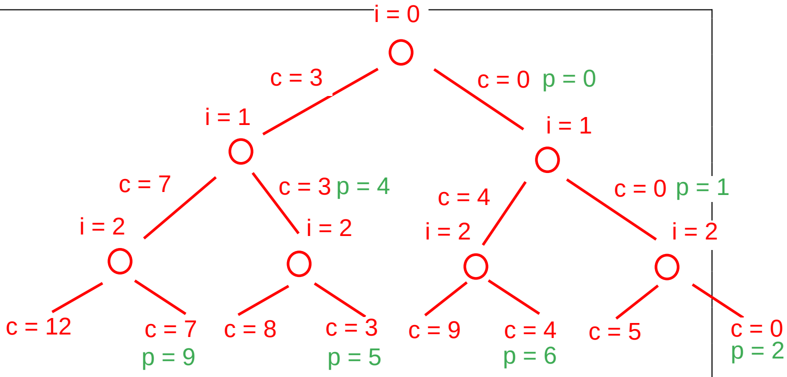
```
int can_deadlock(struct thread *thread) {

     struct thread *next;
     if (curthread == thread) {
         return 1;
     }
     for (next = thread->wait; next != NULL; next = next->wait) {
         if (next == curthread) {
             return 1;
         }
     }
     return 0;

}
```

Question 5. Fork Me [10 MARKS]

```
int main()
{
    int c = 0;
    for (int i = 0; i < 3; i++) {
        if (fork() == 0) {
            printf("%d␣", c + i);
            fflush(stdout);
        } else {
            c = c + i + 3;
        }
    }
    return 0;
}
```



Part (a) [2 MARKS] In the code shown above, how many times is `fork()` invoked?

7 times. In the tree diagram shown above, each circle is a fork.

Part (b) [4 MARKS] Circle all the output sequences shown below that are valid program output. Marks will be deducted for incorrect answers. Hint: Make a diagram that shows program execution, and label the variable values.

[✓ 1.] 0 1 4 6 5 9 2

2. 0 1 5 4 9 6 2

3. 0 4 2 6 1 5 9

[✓ 4.] 9 4 0 5 1 2 6

[✓ 5.] 4 0 1 6 9 2 5

6. 6 4 9 5 0 1 2

Note that only the child process prints values. These values are shown in green above. The only dependencies that exist are 0 -> 1 -> 2, 0 -> 6, 4 -> 5.

Part (c) [4 MARKS] Suppose the system is short on memory and can only run 4 processes. As a result, after 4 processes are created, all fork() invocations will fail, even if the previously created processes terminate. Circle all the output sequences shown below that are valid program output. Marks will be deducted for incorrect answers.

1. 0 3 7

[✓ 2.] 4 0 1

3. 5 4 0

[✓ 4.] 9 4 0

[✓ 5.] 0 1 6

6. 9 0 1

With 4 processes, only 3 fork() invocations will succeed.
0 3 7: 3 and 7 are incorrect outputs.
5 4 0: will not work because 4 needs to be printed before 5.
9 0 1: this would have required 4 forks to succeed (see diagram above).

Question 6. Tagged TLB and Page Table [14 MARKS]

An early 16-bit processor uses a single-level page table with 256-byte pages. The maximum amount of physical memory it can support is 2^{16} bytes. The TLB and the PTE have the following format:

TLB:

VPN	PFN	PID	–	O	R	V

PTE:

PFN	PID	S	O	–	V

The PTE size is 2 bytes. O = readonly bit, R = referenced bit, V = valid bit, and S = currently in swap (PFN should then be interpreted as the swap location). The OS looks at the swap bit only when the valid bit is 0.

Part (a) [1 MARK] Calculate the maximum number of processes whose page table entries may exist in the tagged TLB at a time.

Max physical memory size = 2^16, Page size = 256 = 2^8

Maximum number of frames = 2^16/2^8 = 2^8

PFN size = 8 bits

PTE size is 2 bytes = 16 bits

So PID size is 4 bits, Maximum number of processes = 2^4 = 16

Part (b) [2 MARKS] What is the maximum size of the swap area that is supported?

PFN: 8 bits, Page size = 8 bits

256 * 256 = 65536 = 64KB

Part (c) [2 MARKS] This processor supports a limited amount of physical memory. As a result, the page tables of four processes are packed in a frame. For example, the page tables of Process 4, 5, 6, 7 are located in one frame, and Process 5 uses the second chunk as its page table. What is the maximum amount of virtual memory that is available to each process?

Page size = 256 bytes

Page table size per process = 256/4 = 64 bytes

PTE size = 2 bytes

#PTE per process = 64/2 = 32

Virtual address space size = #PTE * page size = 32 * 256 = 2^13 = 8KB

Part (d) [1 MARK] If the processor uses 64 bytes for the TLB, what is the maximum number of TLB entries that can be stored in the TLB?

TLB entry size = 2 bytes (PTE) + minimum nr of bits for VPN

minimum nr of bits for VPN = 5 (we have a maximum of 32 pages per process, see Part (c))

minimum TLB entry size = 21 bits

maximum number of TLB entries = 64 * 8/21 = 512/21 = 24

Part (e) [1 MARK] The following frame is used to store the page table for processes 0 to 3. Use an outline to show the location of the page table for process 0.

	0	1	2	3	4	5	6	7
0x00	0x78A8	0x5401	0xBC01	0x9401	0xF9EA	0xDB08	0x5404	0x2652
0x10	0x5400	0x5402	0x030E	0xB7F8	0xB00B	0x3E01	0x550F	0x0401
0x20	0xBC16	0xAEE2	0xB02C	0x5803	0x0D38	0xAC03	0xB4F4	0xCFBA
0x30	0x2764	0x0B4E	0x4C86	0xE805	0xBCCA	0xEE01	0xC607	0xC909
0x40	0xBA15	0x5419	0xBC13	0x3C19	0xEF28	0x816C	0x24CA	0xAB11
0x50	0x34F8	0x5400	0xB63C	0xFE13	0x3A11	0x3B2C	0x59B2	0xE619
0x60	0xBC1C	0x441F	0x15F2	0xBD6C	0xAD46	0x1413	0xE65A	0x4017
0x70	0x9B11	0x4715	0xF911	0x0618	0xF3FC	0xC722	0x9000	0x193E
0x80	0xC36E	0x5400	0xBC32	0xBC10	0x8323	0x5E1C	0xEC21	0x0260
0x90	0xD623	0x540E	0x8021	0xB9EA	0x7C25	0xF894	0x3064	0x4623
0xA0	0xBC2C	0xBC12	0x9659	0xA574	0x20C0	0x6D2C	0xB552	0x4B88
0xB0	0x72EA	0xD48A	0x3821	0x4F27	0x2BE8	0x287E	0x6421	0x98E2
0xC0	0x1657	0x540A	0xBC02	0x9FE7	0x1CF2	0x89CC	0x70F7	0x34DA
0xD0	0xBC16	0x5406	0x0A0A	0xECE3	0x4415	0x1337	0xF1D0	0xBEEF
0xE0	0x115A	0xDEAD	0xB0D1	0xFEED	0x9EF9	0x1C8D	0x0430	0xC0C6
0xF0	0x5FBF	0x5408	0x563F	0x3DF5	0x8033	0x5400	0x0CF4	0x5400

These are the 32 PTE for process 0

Part (f) [5 MARKS] Using the table above, translate the virtual addresses shown below for process 0 to their corresponding physical addresses. For each address, show the corresponding PTE and the physical address. If the virtual address is invalid, write FAULT under PTE. If the PTE is not valid, write INVALID under physical address. If the page is in swap, write SWAP X, where X is its disk location.

Virtual address	PTE	Physical address
0x0100	0x5401	0x5400
0x15B4	0xAC03	0xACB4
0x2001	Fault	
0x05F3	0xDB08	Invalid
0x0A47	0x030E	Swap 0x03
0x1EF0	0xC607	0xC6F0

Part (g) [2 MARKS] The table on the left shows the current contents of the TLB. For each operation shown on the right, indicate whether a TLB hit or a TLB miss occurs. Assume that the TLB contents are not modified on a hit or a miss.

TLB
06 F4 41
03 94 03
07 26 01
1B E8 05
19 47 16
00 BA 15
18 9B 11
1D EE 01

VPN PFN Pid

Operation		TLB hit or miss
Process 1 reads from address	0x0343	Miss, no TLB entry for proc 1
Process 1 reads from address	0x1938	Miss, TLB entry is invalid
Process 0 writes to address	0x1D49	Hit
Process 0 writes to address	0x1B88	Miss, wrong protection

Question 7. RAID [6 MARKS]

Many RAID devices now ship with the following options:

1. RAID 0 - data striped across all disks
2. RAID 1 - each disk is mirrored
3. RAID 5 - striped parity

Assume a RAID system of 8 disks. Disk throughput in a RAID system is specified in the number of IO requests (reads or writes) per second. Assume each disk can serve 100 requests/second.

Part (a) [2 MARKS] For each RAID level, how much usable storage does the system receive?

Level	Answer
RAID0:	8 disks
RAID1:	4 disks
RAID5:	7 disks

Part (b) [2 MARKS] Assume a workload consisting only of small reads, evenly distributed across disks. Assuming that no verification (checking for errors) is performed on reads, what is the throughput of each RAID level? You may briefly explain your answer.

Level	Answer	Explanatation
RAID0:	800 req/s	All disks can operate concurrently
RAID1:	800 req/s	All disks can operate concurrently (the mirrors can be used to access different data)
RAID5:	700 req/s	All disks (other than the parity disk) can be accessed concurrently

Part (c) [2 MARKS] Assume a workload consisting only of small writes, evenly distributed across disks. Again, calculate the throughput of each RAID level. You may briefly explain your answer.

Level	Answer	Explanatation
RAID0:	800 req/s	All disks can operate concurrently
RAID1:	400 req/s	Each write has to be mirrored
RAID5:	200 req/s	Each write has perform (1 read + 1 write) from the disk being written and from parity disk

Question 8. In-Place versus Log-Structured File Systems [18 MARKS]

In a Unix file system, the inode structure contains 12 direct pointers, 1 singly-indirect, 1 doubly-indirect, and 1 triply-indirect pointer. Assume that the access or modified time in the inode are not tracked by the file system, and the file system does not perform journaling. Each pointer is 32 bits wide, and that file system block size is 4KB.

The file system data is stored on a disk with the following characteristics: seek time = 4 ms, rotational latency = 4 ms, block transfer time = 0.1 ms.

Note: 1KB = 1024 bytes, 1MB = 1024KB, 1GB = 1024MB.

Part (a) [1 MARK] How many pointers will fit in an indirect block?

4096 bytes / 32 bits = 4096/4 = 1024 pointers

Part (b) [6 MARKS] How many blocks of each type shown below are created or updated when a new 4GB file is created in the root directory of the Unix file system? Assume that the file is stored contiguously on disk. You may write your answer in a simple form or as a multiple of a power-of-2 expression. The indirect block shown below includes all types of indirect blocks. Show your calculations below.

inode block = 1	block bitmap block = 33	inode bitmap block = 1
data block = 2^20	indirect block = 1025	directory block = 1

inode block, inode bitmap block, directory block = 1 block each, for the new file being created

data blocks = 4GB/4KB = (4 * 2^30) / (4 * 2^10) = 2^20

Direct blocks can store 48KB of data
One single indirect block can store 1024 * 4KB = 4MB of data
One double indirect block can store 1024 * 1024 * 4KB = 4GB of data (this requires 1024 single indirect blocks as well)

The last single indirect block in the double indirect block is not used because of the single indirect in the inode structure. Therefore the number of indirect blocks = 1 single indirect + 1 double indirect + 1023 single indirect = 1025

Total number of dynamic blocks allocated = 2^20 + 1025 + 1 (data, indirect and directory)
Nr of bits in block bitmap = 4 * 1024 * 8 = 2^15 = 32768
Assuming mostly contiguous allocation, # of block bitmap blocks = (2^20 + 1026)/2^15 = 32 + 1 = 33

Part (c) [2 MARKS] How many blocks of each type are created or updated when we modify the last byte of the 4GB file in the Unix file system?

inode block = 0	block bitmap block = 0	inode bitmap block = 0
data block = 1	indirect block = 0	directory block = 0

Part (d) [3 MARKS] Suppose that a log-structured file system uses the same inode structure as the Unix file system. How many blocks of each type are updated when we modify the last byte of an existing 4GB file? Assume that the log-structured file system does not use bitmap blocks.

inode block = 1	inode map block = 1	checkpoint block = 1
data block = 1	indirect block = 2	directory block = 0

Part (e) [2 MARKS] Say four (4) 4 GB files exist in the Unix file system. Based on Part c), calculate the amount of time needed to modify the last byte of all the four 4GB files in the file system. Assume blocks of different types are stored non-contiguously on disk. Show your calculations.

There will be a seek to modify each file.
Each file access will require 4 + 4 + 0.1 = 8.1 ms
Total = 32.4 ms

Part (f) [2 MARKS] Say the same four (4) 4 GB files exist in the log-structured file system. Based on Part d), calculate the amount of time needed to modify the last byte of all the four 4GB files in the log-structured file system. Assume that the checkpoint block is written after each file operation. Show your calculations.

5 blocks (inode, inode map, data, indirect) are written to the log and then a checkpoint block is written per file. This will require two seeks, one to get to the log, and one to get to the checkpoint. Each file access will require 8 (seek to log) + 0.5 (write 5 blocks to log) + 8 (seek to checkpoint) + 0.1 (write to checkpoint) = 16.6 ms.

Total = 16.6 * 4 = 66.4 ms.

Part (g) [2 MARKS] How much time is needed if the checkpoint block is written once after all the four file operations? Show your calculations.

Total = 8 (seek to log) + 5 * 0.1 * 4 (write to the log for all the four files) + 8 (seek to checkpoint) + 0.1 (write to checkpoint) = 8 + 2 + 8 + 0.1 = 18.1 ms

Question 9. File System Consistency [10 MARKS]

A journaling file system allows file system operations to be performed failure atomically, i.e., an operation appears to either have been fully done or not done at all, in the presence of operating system crashes or power failures.

Joe, a journaling file system designer wishes to optimize his journaling file system. His current file system journals all metadata and data blocks of the file system to the journal. Thus each file system write becomes two writes. To optimize performance, Joe considers avoiding journaling of the block bitmap and inode bitmap blocks. Instead, he plans to write these blocks directly to the file system.

Part (a) [4 MARKS] Describe clearly and precisely how Joe should perform file system recovery after a crash so that the bitmaps can be made consistent.

The inode bitmap encodes the inodes that are currently allocated. These can be obtained by traversing the entire file system tree, while keeping track of every inode number present in every directory entry.

The block bitmap encodes the blocks that are currently allocated. These can be obtained by traversing the entire file system tree, while keeping track of every allocated block in each inode.

Many answers mentioned that the bitmaps could be made consistent by looking at the blocks that were logged in the journal. The problem with this is that a bitmap could have been modified, nothing could have been logged and then a crash occurs. In this case, it is not possible to fix the bitmaps.

Part (b) [2 MARKS] Should the recovery operations described in Part a) above be performed before or after journal recovery? Explain why.

The journal recovery operations should be performed first so that all the other blocks become consistent and then the bitmaps can be recovered correctly.

Part (c) [2 MARKS] Can the bitmaps always be recovered to a consistent state? If so, why? If not, why not?

They can be recovered to a consistent state because all other blocks are consistent, and Part (a) recreates all the bitmaps based on the consistent state of all the blocks.

Part (d) [2 MARKS] Is it worthwhile for Joe to implement this journaling optimization? Clearly justify your answer.

This optimization is not worthwhile because the total number of bitmap blocks are relatively small compared to the rest of the blocks. So this optimization will not reduce journaling costs much. However, recovery requires a full file system scan, which is very expensive.

[Use the space below for rough work.]