Information Technology University of the Punjab
FoE Final Examination – Spring 2025
SE301T – Operating Systems (A)

Name: ________________Solution________________________ Roll No.: ________________________

Time allowed: 180 minutes Maximum marks: 75
Date & day: 3rd June, 2025 (Tuesday)

Instructions

a. This exam will assess your CLOs as per OBE. The CLOs are mentioned below.
b. This is a **CLOSED BOOK** and **CLOSED NOTES** exam.
c. The exam has **FIVE (5)** questions (labeled as Question 1, Question 2, …), which have multiple parts (labeled as a., b., …) and subparts (labeled as i., ii., iii., …).
d. Use of a calculator is allowed, but you must use your own calculator. **Do not borrow calculators from others.**
e. Answer all questions in the given space. **No extra sheets will be provided.**
f. **If you are found cheating (using unfair means, unauthorized material or unauthorized equipment) or helping others cheat, your exam will be cancelled immediately and disciplinary action will be taken.**

CLOs

1. **Understand** fundamental operating system abstractions such as processes, threads, files, semaphores, IPC abstractions, shared memory regions, etc.
2. **Develop** important algorithms and services provided by the operating systems: Scheduling, memory management, ls, cat, zip, etc.
3. **Develop** multi-threaded applications using the learnt parallel programming concepts

---

Q1 – CLO1	Q2 – CLO2	Q3 – CLO2	Q4 – CLO2	Q5 – CLO3	Total
/ 10	/ 15	/ 15	/ 15	/ 20	/ 75

Instructor: Umair Shoaib Instructor's signature: ______________________
Teaching Assistant: Fatima Ehsan

KGH's signature: ______________________

Question 1 – CLO 1 (Cognitive level 1 i.e. knowing, PLO1) **[10]**

a. **[7]** Have a look at the following program and answer the questions below:

```
int main() {
    int pipe1[2];
    int pipe2[2];
    pipe(pipe1);

    pid_t rc = fork();
    if (rc == 0) {
        close(pipe1[1]);
        dup2(pipe1[0], STDIN_FILENO);
        close(pipe1[0]);
      // -w option of wc counts words
        execlp("wc", "wc", "-w", NULL);
        perror("execlp");
        exit(1);
    }

    pipe(pipe2);

    pid_t rc2 = fork();
    if (rc2 == 0) {
        close(pipe1[0]);
        close(pipe1[1]);
        close(pipe2[1]);
        dup2(pipe2[0], STDIN_FILENO);
        close(pipe2[0]);
      // -c option of wc counts characters
        execlp("wc", "wc", "-c", NULL);
        perror("execlp");
        exit(1);
    }

    close(pipe1[0]);
    close(pipe2[0]);

    const char *msg = "Hi there!\n";
    write(pipe1[1], msg, strlen(msg));
    write(pipe2[1], msg, strlen(msg));

    close(pipe1[1]);
    close(pipe2[1]);

    wait(NULL);
    wait(NULL);
    return 0;
}
```

i. [1] How many total processes are created by this program, including the original parent process?

3 – one parent, two children (rc and rc2)

ii. [1] The program has two pipes, pipe1 and pipe2. Which of these two pipes is not shared among all processes in the program?

The pipe pipe2 is not shared by the first child process as this child is created before the pipe2 is created.

iii. [3] Describe how each process in the program uses the two pipes. For each process, specify which ends of the pipes are open or closed, and explain what data is being read from or written to the pipes.

Child1 (rc):

Closes write end of pipe1
Duplicates read end of pipe1 to STDIN_FILENO
Wc called inside this child receives its input from read end of pipe1.

Child2 (rc2):
Closes read and write ends of pipe1
Closes write end of pipe2
Duplicates read end of pipe2 to STDIN_FILENO
Wc called inside this child receives its input from read end of pipe2.

Parent
Closes read ends of both pipes pipe1 and pipe2
Writes “Hi there!” to the write ends of both pipes, which means that “Hi there!” will be received at read ends of pipe1 and pipe2 inside Child1 and Child2 respectively.

iv. [2] What would the output of this program be?

10

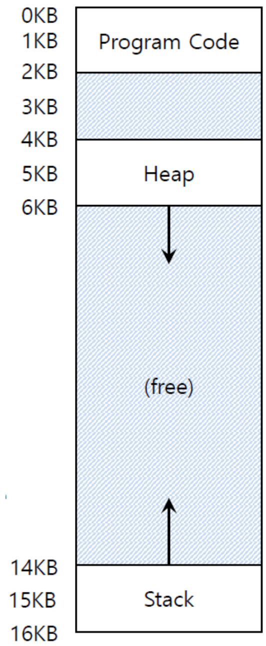
2

b. **[3]** Give three differences between a hard link and a symbolic (soft) link.

Hard link	**Soft link**
Points to the same inode	Points to a different inode that has the filename (path)
If the original file is deleted, a hard link still points to the contents of the file	If the original file is deleted, a soft link becomes a dangling pointer
Its Linux command is: ln file [hardlink]	Its Linux command is: ln –s [file] [softlink]
Link count of the file increases by 1 when a hard link is created	Link count does not increase
Directories cannot have a hard link	Soft link can be of directories
Hard link cannot be accessed across different file systems	Soft links can be accessed across different file systems

Question 2 – CLO2 (Cognitive level 3 i.e. applying, PLO2) **[15]**

a. **[4]** The adjacent figure shows a virtual address space of size 16KB. The base and size registers along with direction of growth and top bits for identification of each segment are given below:



Segment	Base	Size	Bits	Grows positive?
Code	32K	2KB	00	1
Heap	34K	2KB	01	1
Stack	28K	4KB	11	0

Determine the address translation for the virtual address `0x3800` (represented in hexadecimal) and also mention the segment this address belongs to.

0x3800 in decimal: 14336

0x3800 in binary: 00<u>**11**</u> **1000 0000 0000 (lies in stack region as two MSBs of 14-bit representation are 11)**

So physical address is equal to: 28K – (16K – 14336) = 28672 – (2048) = **26624**

26624 lies inside 24K – 28K (range of stack addresses) so it is a valid translation.

b. **[11]** Consider an address space described as follows:

Page size: 32 bytes
Virtual address space size: 32KB or 1024 pages
Levels of paging: 2
Physical memory size: 4KB or 128 page frames
Size of Page Directory Entry (PDE) and Page Table Entry (PTE): 1 byte
In both PDE and PTE, the Most Significant Bit (MSB) indicates validity (0 for invalid, 1 for valid). The rest of the bits of the PDE and PTE are for the PFN.

Page Directory Base Register (PDBR): 122 (decimal)

A complete dump of the physical memory is provided at the page 17 of this exam. You may remove this page so that you can look at the entire memory for solving this question.
Give answers of the following questions:

i. [2] Determine the number of PDEs in this address space that are valid.

28

ii. [2] Determine how much memory space this page table would take. Note that each valid PDE holds the PFN of the level-2 page table in which there is at least one valid PTE. In 2-level page tables, only pages holding at least one valid PTE are stored in memory. You already have the valid PDEs from your answer in the above question. You can work from there the total space required to store the complete page table.

So, total space required will be 28 pages for level-2 page tables and one page for the Page Directory.

So a total of 29 pages = 29*32 bytes = 802 bytes.

iii. [7] Determine the physical address corresponding to the following virtual addresses. In case a virtual address does not lie in a valid page frame, you should mention that the translation is invalid. Also required are a few other values which you will have to find on your way to reaching the physical address. Lastly, you are expected to provide the contents at the physical address.

- Virtual address: 0x11e2

Offset: 0x02____________________

Page Directory Index: (Decimal) 4____________________

Page Table Index: (Decimal) 15____________________

Page Directory Entry: 0xff____________________

Page Table Entry: 0x83____________________

Physical Address: 0x62____________________

Contents at the physical address: 0x14____________________

- Virtual address: 0x5b92

  Offset: 0x12____________________

  Page Directory Index: (Decimal) 22____________________

  Page Table Index: (Decimal) 28____________________

  Page Directory Entry: 0xa2____________________

  Page Table Entry: 0x7f____________________

  Physical Address: 0xInvalid____________________

  Contents at the physical address: 0xInvalid____________________

Question 3 – CLO2 (Cognitive level 3 i.e. applying, PLO2) **[15]**

a. **[8]** Consider a process which executes the following x86 assembly code:

```
0x18      movl    $0x0$, %eax           ; move 0 to eax register (eax is the iterator)
0x1C      movl    $0x41C$, %edi         ; move address 0x41C to edi register
0x20      movl    $0x0, (%edi, %eax, 4) ; store 0 at the location pointed to by edi + 4 * eax
0x24      jmp     0x38                  ; jump to instruction at 0x38
          …
0x38      movl    $0x818$, %edi         ; move address 0x818 to edi register
0x3C      movl    $0x0, (%edi, %eax, 4) ; store 0 at the location pointed to by edi + 4 * eax
0x40      incl    %eax                  ; increment iterator eax by 1
0x44      cmpl    $0xA, %eax            ; compare eax with 10 (0xA)
0x48      jne     0x1C                  ; if eax ≠ 10, jump back to 0x1C instruction
```

Note that the virtual address of each instruction is given at the beginning of each line. Now, suppose that:

- The size of the address space is 32KB.
- Paging is used to virtualize the address space of this process.
- The page size is unusually small, at just 32 bytes.
- The system has a TLB of only four (4) entries
- TLB uses the Least Recently Used (LRU) replacement policy.
- The TLB is initially empty.

You need to update the state of the TLB for 10 instruction executions starting with the first instruction at 0x18. Remember that for the execution of the instructions of the type at address 0x20 and 0x3C, there have to be made two memory accesses: one to access the address of the instruction, second to access the address in the instruction operands, such as (%edi, %eax, 4). For each instruction execution, you need to mention if the TLB entries for the memory accesses in that instruction were Hit (H) or Miss (M).

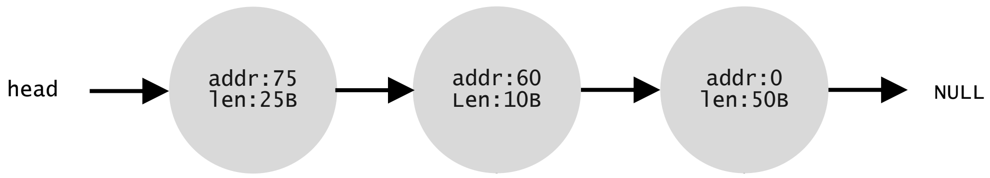
For the first two instructions, the TLB state has been filled for you. Here, the addresses 0x18 and 0x1C had to be accessed. Both of these addresses are in page 0 of the address space (VPN bits of the virtual address tell you the page number). Now, when first instruction is accessed, the TLB is empty and it is considered a miss. Thereon, instruction's VPN and corresponding PFN and related bits are entered into the TLB (for simplicity, we have skipped the PFN and permission bits etc. here). When the second instruction is accessed, the TLB already has the translation for VPN, so it is a hit.

Fill the TLB state for the remaining 8 executions in the tables below:

	Instruction 1		**Instruction 2**		**Instruction 3**		**Instruction 4**		**Instruction 5**	
TLB Entry	**VPN**	**H / M**	**VPN**	**H / M**	**VPN**	**H / M**	**VPN**	**H / M**	**VPN**	**H / M**
0	0	M	0	H	0		0		0	
1	-		-		1	M	1	H	1	H
2	-		-		32	M	32		32	
3	-		-							

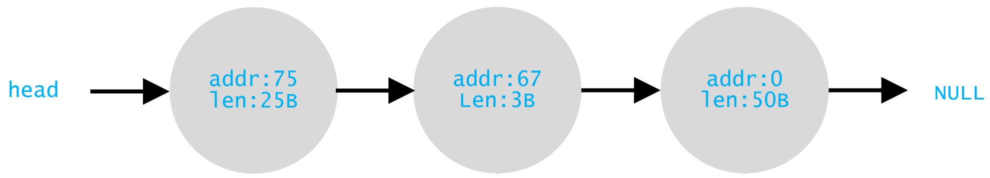
	Instruction 6		Instruction 7		Instruction 8		Instruction 9		Instruction 10	
TLB Entry	**VPN**	**H / M**	**VPN**	**H / M**	**VPN**	**H / M**	**VPN**	**H / M**	**VPN**	**H / M**
0	0		2	M	2	H	2	H	2	
1	1	H	1		1		1		1	
2	32		32		32		32		0	M
3	64	M	64		64		64		64	

b. **[4]** Consider a free list represented as follows:

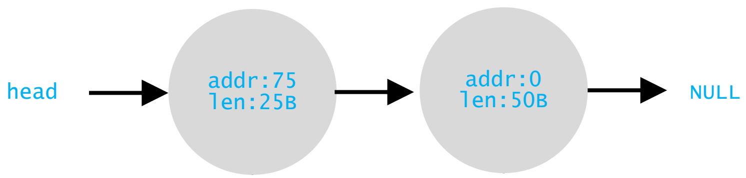


Suppose that the "best-fit" strategy of memory allocation is used. Also, assume that the metadata does not take any space inside or outside the allocated region. Draw final representation of the free list after each of the following events that occur in the given order:

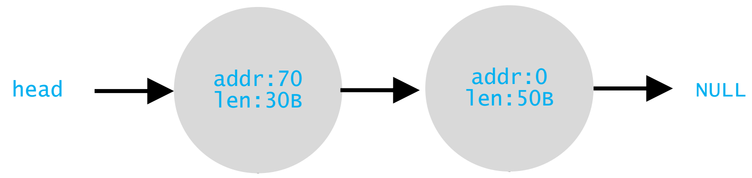
i. [1] A request of 7 bytes



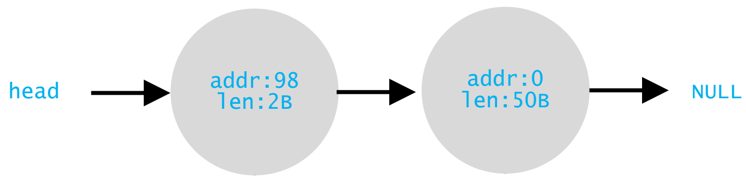
ii. [1] A request of 3 bytes



iii. [1] 5 bytes of memory freed at addr 70



iv. [1] A request of 28 bytes



c. **[3]** How is “splitting” and “coalescing” of memory handled by the buddy allocator? Explain with the help of one example for each of splitting and coalescing.

Splitting: When there is a request of x size of memory, buddy allocator splits free memory into equal halves that have size in power of two, until further splitting causes segment size to be less than the requested allocation “x”.

For example, if the request is of 13 bytes, buddy allocator would divide the free memory, say 64 bytes, into two halves of 32 bytes, then one of the 32 byte section into two 16 bytes. At this point further splitting would cause the section size to be less than requested allocation of 13 bytes, so no further splitting takes place and one of the 16-byte section is provided as the requested allocation.

Coalescing: When a power of two memory section is freed, buddy allocator checks the address and size of its adjacent section. If size is equal and the address varies by only 1 bit, it combines both to form a double-sized memory section.

Question 4 – CLO2 (Cognitive level 3 i.e. applying, PLO2) **[15]**

a. **[10]** Ticket lock uses the atomic fetch-and-add instruction to ensure that the lock is acquired in a FIFO manner. Thus, ticket locks improve the fairness metric in evaluating locks. Provide answers of the following questions related to ticket locks.

i. [5] Provide C code for the implementation of a ticket lock (initialization, acquire and release functions) using the fetch-and-add instruction. The C-like pseudocode of the fetch-and-add instruction and function prototypes are provided to you:

```
int FetchAndAdd ( int *ptr ) {
        int old = *ptr;
        *ptr = old + 1;
        return old;
}

typedef struct __lock_t {
        int ticket;
        int turn;
} lock_t;

void lock_init (lock_t *lock) {

        lock->ticket = 0;
        lock->turn = 0;

}

void acquire (lock_t *lock) {
        int myturn = FetchAndAdd(&lock->ticket);
        while(lock->turn != myturn)
             ; // spin

}

void release (lock_t *lock) {

        FetchAndAdd(&lock->turn);

}
```

ii. [5] Three threads running concurrently attempt to acquire a ticket lock in this sequence: T1 first, followed by T3, and then T2. The sequence of their operation is shown in the first three columns of the table below with time progressing downwards. You are required to fill in the values of two fields of the ticket lock, ticket and turn, at each step during execution. Assume that the lock is freshly initialized before T1 calls acquire, and the lock has not been acquired or released before this point.

			Ticket	**Turn**
T1 calls acquire			1	0
T1 acquires the lock and enters critical section			1	0
		T3 calls acquire	2	0
		T3 waits	2	0
	T2 calls acquire		3	0
	T2 waits		3	0
Critical section completes, T1 releases the lock			3	1
		T3 acquires the lock and enters its critical section	3	1
		Critical section completes, T3 releases the lock	3	2
	T2 acquires the lock		3	2
	Critical section completes, T2 releases the lock		3	3

b. **[5]** A company's payroll system uses multiple threads to update employee salary records in a shared database. The following simplified C-style code shows the logic used to add a bonus to an employee's record:

```
struct employee {
    char name[32];
    int salary;
};

void give_bonus(struct employee *emp, int bonus) {
    int current_salary = emp->salary;
    current_salary += bonus;
    emp->salary = current_salary;
}
```

The HR department runs this function concurrently from two different threads (without using any locks) for the same employee: one to give a performance bonus of 20,000 PKR, and another to give an Eid bonus of 30,000 PKR. Assume that the initial salary of the employee is 100,000 PKR. The final salary of the employee for this month should then be 150,000 PKR, but due to the absence of a synchronization mechanism, the code *can* return an incorrect final salary. Answer the following questions pertaining to this problem:

i. [1] What is a possible incorrect final salary that could occur due to a race condition?

120000 (can be 130000 as well)

ii. [3] Explain how the race condition occurs in this scenario.

In the case 120000 is returned:
Performance bonus thread starts: updates local variable current_salary to emp->salary (100000 here).
Adds 20000 to current_salary.
Performance bonus thread preempted.
Eid bonus thread starts: updates local variable current_salary to emp->salary (still 100000).
Adds 30000 to current_salary.
Updates emp->salary to 130000.
Eid bonus thread exits.
Performance bonus thread resumes.
Updates emp->salary to 120000.
Performance bonus thread exits.

iii. [1] Suggest a solution to prevent this race condition.

By using a lock around accessing emp->salary inside thread functions.

Question 5 – CLO3 (Cognitive level 3 i.e. applying, PLO3) **[20]**

a. **[5]** Consider the following multi-threaded program:

```
int x = 3;
int a = 3;

void *print_int(void *arg) {
    int k = *((int *)arg);
    printf("k = %d\n", k);
}

void main() {
    pthread_t threads[x+1];
    while(x > 0) {
            pthread_create(&threads[x], NULL, print_int, &a);
            x--;
    }
    // code to join all threads
}
```

i. [2] What will the output of this program be?

k = 3
k = 3
k = 3

ii. [3] What would the output be if the argument "&a" of each thread in the pthread_create function is replaced with "&x"?

The output cannot be determined, as x may be 0, 1, 2 or 3 depending on when a thread runs. So we may observe any possible combinations, including duplicate values. For example:

k = 1
k = 1
k = 3

k = 0
k = 0
k = 0

k = 3
k = 2
k = 1

b. **[5]** An old man is expecting N > 2 visitors at his house today. Due to his old age, he does not wish to get up and open the door every time a visitor comes. Instead, he wishes that all N visitors, even though they may arrive at different times to his door, wait for each other and enter the house all at once. The old man and the visitors are represented by threads in a multithreaded program.

Given below is the pseudocode for the old man's thread, where the old man waits for all visitors to arrive, then calls openDoor(), and signals a condition variable once.

```
// old man
lock(m)
while(visitor_count < N)
      wait(cv_oldman, m)
openDoor()
signal(cv_visitor)
unlock(m)
```

You are required to provide the corresponding pseudocode for the visitor threads. The visitors must wait for all N of them to arrive and for the old man to open the door, and must call enterHouse() only after that. You must ensure that all N waiting visitors enter the house after the door is opened. You must use only locks and condition variables for synchronization.

The following variables are to be used in this solution: lock m, condition variables cv_oldman and cv_visitor, and integer visitor_count (initialized to 0). You must not use any other variables in the visitor's code for synchronization.

Excluded from exam!

c. **[10]** You are required to implement a multithreaded C program that computes the factorial of a given number N using two threads. The two threads must alternate turns to perform the multiplication steps for computing the factorial. Thread T1 should always start first. The multiplication pattern should alternate between the two threads as illustrated in the example below (here N = 5):

```
T1: 1 × 2 (T1 always starts first)
T2: 2 × 3
T1: 6 × 4
T2: 24 × 5
```

Only semaphores can be used for synchronization. No other synchronization primitives are allowed. A skeleton code is provided below. Fill in your code where the comments start with "ToDo".

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

#define N 5 // N! is required


int result =    1        ; // ToDo: Initial value of result


int current =   2        ; // ToDo: Initial value of current

sem_t sem_t1;
sem_t sem_t2;

void* thread1_func(void* arg) { // Thread 1
    while (1) {
            // ToDo: Wait for your turn

            sem_wait(&sem_t1);



            if (current > N) {
                    // ToDo: Signal the other thread to finish

                    sem_post(&sem_t2);



                    break;
            }
            // ToDo: Perform multiplication step for T1 and then increment 'current'
            result *= current;

            current++;


            // ToDo: Signal the other thread
            sem_post(&sem_t2);



    }
    return NULL;
}

void* thread2_func(void* arg) {
    while (1) {
            // ToDo: Wait for your turn
```

```
        sem_wait(&sem_t2);

        if (current > N) {
            // ToDo: Signal the other thread to finish
            sem_post(&sem_t1);

            break;
        }
        // ToDo: Perform multiplication step for T2 and increment 'current'

        result *= current;

        current++;

        // ToDo: Signal the other thread

        sem_post(&sem_t1);

    }
    return NULL;
}

int main() {
    pthread_t t1, t2;

    // Initialize semaphores

    // ToDo: Initialize sem_t1 on which T1 waits. T1 should start first, so think about what the initial
value of sem_t1 should be.

       sem_init(&sem_t1, 0, 1); // initial value = 1

    // ToDo: Initialize sem_t2 on which T2 waits. T2 should not start first, so think about what the
initial value of sem_t2 should be.
       sem_init(&sem_t2, 0, 0); // initial value = 0

    pthread_create(&t1, NULL, thread1_func, NULL);
    pthread_create(&t2, NULL, thread2_func, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("Factorial of %d is %d\n", N, result);

    // Destroy semaphores
    sem_destroy(&sem_t1);
    sem_destroy(&sem_t2);

    return 0;
}
```

Physical memory dump for Question 2 (b)

Byte:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
page 0	11	0d	16	1b	7	13	4	0d	0a	19	1c	1a	10	16	14	10	4	8	1d	12	3	0b	4	3	0b	9	0	0f	1c	0e	0c	0b
page 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 2	1b	0c	1e	19	12	0d	11	1a	0a	4	1b	13	0b	0c	3	11	18	0f	15	16	0a	13	1e	16	5	14	13	18	15	1c	1e	6
page 3	5	3	14	0a	1b	7	1d	9	12	1c	15	1c	15	1	1b	12	9	5	1a	12	1d	17	1d	11	15	12	8	1d	9	18	1d	14
page 4	0e	0b	0b	17	0	8	1d	0e	0	13	14	1c	1b	1d	8	9	15	0a	0b	1	6	4	0b	15	6	8	2	0c	12	12	14	10
page 5	3	0d	13	0d	5	0f	6	0f	16	9	0d	13	10	18	1b	1d	5	0e	3	15	0c	1a	0e	18	0	13	0c	19	3	2	15	0a
page 6	6	6	10	1	2	2	16	1e	17	0f	4	18	4	2	0f	1a	9	1a	0a	1d	16	1a	1e	19	6	0f	10	16	7	11	8	5
page 7	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	d2	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 8	7f	7f	d8	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 11	0e	0f	15	1d	18	10	15	10	1e	12	12	0c	0c	17	0b	1c	16	1d	15	11	0b	0	14	17	16	17	1	15	8	0c	1a	0c
page 12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 13	14	6	16	0c	19	11	1b	0d	0c	1c	15	0d	1b	16	13	0	1b	3	0	0a	6	7	0f	0f	15	1a	15	1a	0d	4	16	1e
page 14	7f	7f	7f	7f	7f	7f	7f	8d	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	e8	7f	d9	7f	7f	7f	7f	7f	7f	7f	7f	ca	7f	7f
page 15	14	5	1e	0e	7	13	1	14	1a	3	7	8	6	13	19	0e	15	1c	19	0b	1b	5	19	2	9	3	11	3	12	5	14	0f
page 16	14	8	0	0d	19	4	1d	0f	4	3	0b	0e	5	2	0f	7	0d	0d	15	11	7	5	1a	1	3	2	19	10	1c	13	0a	0d
page 17	0	1	6	18	18	18	15	1e	7	15	0d	1d	0d	16	18	1d	17	0a	14	18	1d	7	16	16	4	3	11	12	0a	13	4	11
page 18	17	0b	10	8	1	1e	14	15	13	1	19	1a	7	1a	1d	5	1c	1	17	1e	4	0d	0	10	1	0f	5	5	14	7	4	8
page 19	7	1b	0e	0d	17	0a	7	3	9	7	16	0b	15	8	2	9	0c	1c	6	6	4	7	1d	1c	7	1	0f	12	3	0c	17	0
page 20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 21	19	6	16	0e	0f	2	1b	0	1a	0	0b	0d	12	8	1e	0a	6	16	14	12	1d	1b	9	1b	1b	1	0f	1a	0c	0a	0f	1a
page 22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 23	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	c6	7f	7f	7f	7f	7f	7f	bb	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	86	7f
page 24	7f	7f	7f	7f	fc	7f	7f	7f	7f	7f	7f	f4	7f	7f	7f	7f	7f	fd	7f	7f	ba	7f	7f	f9	7f	7f	7f	7f	7f	7f	7f	7f
page 25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 28	9	0d	9	0b	1a	0	1c	1c	13	17	16	0d	5	3	12	15	16	1e	9	12	8	1a	2	12	1a	7	1e	0c	1e	9	1c	0b
page 29	0c	1	0a	11	4	1c	1c	0f	1c	15	10	4	1d	1c	0e	0b	2	13	1a	0c	8	13	18	5	8	13	1	19	9	19	0e	5
page 30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 31	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	b1	7f	7f
page 32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 33	18	13	9	14	16	0b	2	0e	6	5	0c	2	1b	8	1b	16	0	1b	19	10	1b	0e	0	4	9	1a	19	3	0a	3	7	0a
page 34	7f	7f	b4	7f	b7	7f	7f	7f	eb	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	d7
page 35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 37	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	cf	7f
page 38	7f	7f	7f	7f	7f	b9	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	91	7f	7f	7f	7f	7f	7f
page 39	7f	7f	92	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	ae	7f	7f	7f	7f	7f	ef	7f	7f	7f	7f	7f	7f	7f	82	7f
page 40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 41	1c	0b	1b	0f	13	0e	0c	1b	12	5	9	0c	11	0b	8	9	1e	16	1a	1b	4	10	16	0f	10	13	13	5	1	8	1	9
page 42	7f	7f	7f	ea	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	ac	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 43	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	9c	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 44	0b	5	1	10	18	14	6	15	1	0c	19	1c	0e	1a	0f	10	12	1e	17	0e	16	13	0a	18	19	12	1d	0	0f	13	0a	5
page 45	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	e5	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	8b	7f	7f
page 46	1c	13	10	0d	0e	0	11	1d	3	1	13	19	0d	2	13	14	7	19	10	18	7	0b	14	14	1a	1d	16	0	0e	14	18	9
page 47	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	90	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 49	17	2	9	12	1a	1a	0a	8	0b	3	8	17	1e	8	2	16	1d	18	0c	17	17	17	0f	1e	18	7	0d	2	0b	19	0f	10
page 50	11	5	0c	1c	11	19	8	1a	18	6	10	15	11	12	7	11	0a	18	0b	1e	11	7	0c	3	16	1	12	9	18	3	15	16
page 51	1e	13	19	1a	1a	0d	8	0f	1b	1b	0e	1c	17	1	13	18	1	0b	14	0f	10	2	8	1a	0a	0e	12	8	0c	0f	1a	1a
page 52	11	15	1	0c	0c	19	18	0d	0c	18	16	1	19	1a	8	14	0e	8	1	10	16	0c	13	5	19	16	0f	3	0	0c	7	1
page 53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 54	7f	7f	7f	7f	7f	7f	d3	7f	7f	7f	7f	7f	7f	7f	7f	7f	a9	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 55	1	1d	0a	3	6	7	1a	5	8	3	0e	6	0a	16	1c	19	1	0b	1a	1	6	0f	0b	17	16	0e	7	7	8	2	14	4
page 56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 57	0a	4	1b	3	2	9	4	7	15	0a	1a	0b	1a	19	8	11	14	5	1	1b	0	5	4	1a	14	0f	0f	1d	6	19	9	8
page 58	5	5	3	0f	1a	1c	17	2	13	7	3	3	17	2	1a	0a	0b	1c	5	0d	1a	13	7	3	19	13	1d	5	8	15	9	1e
page 59	6	0d	1a	1	1d	6	0f	9	1	13	5	0c	19	0d	0a	4	0	19	0f	9	17	10	7	1d	0c	0a	1e	0c	0e	19	14	7
page 60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 61	13	0c	2	0d	0	5	0e	0d	13	12	17	11	18	1	0e	1d	17	19	1e	4	0f	1b	9	10	5	2	1c	16	18	1b	11	1a
page 62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 63	7f	7f	7f	7f	7f	8f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	ee	7f	7f	7f	7f	e4	7f	7f	7f	7f	7f

Byte:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
page 64	10	13	14	14	1b	1e	1e	3	16	0c	8	3	1c	1b	10	0f	6	1c	17	5	11	1e	8	0b	0b	0b	3	13	7	15	0e	1a
page 65	7f	7f	7f	7f	7f	7f	7f	7f	c4	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 66	1a	14	0c	1	0a	0f	0b	0d	7	6	1c	18	0b	19	2	13	8	8	0f	11	9	7	0c	19	1c	1b	16	1d	1	1b	0b	1
page 67	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 68	0f	0b	0f	15	0f	17	0b	13	0c	11	17	3	1	6	7	4	11	0e	0e	17	3	1c	18	7	17	1a	0	0a	16	0a	5	8
page 69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 70	0a	7	0b	0c	8	9	7	9	14	13	13	1c	14	4	0c	10	17	10	2	0e	14	18	0d	6	1	0f	17	3	10	12	11	7
page 71	0d	1d	17	2	2	8	19	18	19	1a	16	11	1a	19	19	1b	10	2	13	15	1d	17	1b	1c	0a	1c	11	0d	0d	0f	0c	16
page 72	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 73	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 74	5	10	7	1b	0	11	1b	13	4	10	19	13	1c	19	0b	3	15	0e	19	0d	16	4	17	1a	19	18	14	5	0a	8	9	12
page 75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 76	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	f1	7f	7f	7f	7f	7f
page 77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 78	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	80	7f	7f	7f	7f	7f	95	7f	7f	7f	7f	7f	7f
page 79	0	0d	1e	1e	15	6	6	4	0d	1d	6	19	11	8	0e	1	13	1c	0a	1e	8	0a	15	6	1	0f	0f	5	5	3	5	4
page 80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 81	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 82	0f	12	1	0f	0	17	0	13	1e	1a	7	10	4	0a	1a	19	0	0b	13	5	18	1b	11	0b	1a	0f	5	8	4	1a	0f	15
page 83	1e	16	11	0b	14	1c	0	0b	8	18	14	0	1b	0c	4	3	2	7	2	15	4	0d	14	0d	9	0e	0e	11	12	9	15	1a
page 84	7f	7f	7f	7f	7f	7f	7f	7f	93	7f	7f	7f	df	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	b3	7f	7f
page 85	19	1	1d	1b	0c	6	16	11	3	12	0f	12	0	10	0d	1a	16	16	11	7	19	8	1d	10	0c	0c	0e	0f	4	0c	1	11
page 86	7f	7f	7f	7f	bd	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 87	18	0f	0e	10	4	0f	7	9	1	4	1	10	7	11	0f	7	3	12	14	15	0f	11	0	9	18	0b	3	0d	19	3	3	0f
page 88	2	11	14	8	1e	3	19	10	12	0d	1	0d	15	17	19	9	0f	3	0d	0d	18	2	1	13	18	2	15	4	1a	13	6	0b
page 89	9	1a	0e	16	1	18	6	0d	0f	0a	7	0d	1	4	4	1a	9	1e	1b	5	1a	1e	0d	0e	14	0f	8	16	6	0b	0	0a
page 90	1d	14	8	14	16	4	18	12	12	14	13	4	4	0	9	6	12	3	4	10	18	0f	1	1c	11	1c	17	1c	19	17	1a	14
page 91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 92	0f	0a	1e	0a	10	1b	1e	0d	0b	13	10	10	11	5	5	11	7	0e	1	0b	5	0f	1c	5	6	0a	8	2	2	15	1a	0
page 93	7f	7f	7f	f7	7f	7f	7f	7f	7f	7f	7f	c2	7f	7f	7f	7f	7f	7f	7f	7f	f8	7f	7f	7f	7f	b2	7f	7f	7f	7f	7f	7f
page 94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 95	8	16	0d	1c	6	7	9	2	3	1a	10	18	17	13	0c	1b	9	1b	2	12	1d	10	0b	0e	0d	5	0d	6	0f	10	11	15
page 96	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	da	7f	7f	7f	7f	7f	7f	a1	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 97	12	17	3	6	1d	16	0d	1e	1	12	4	9	6	0c	1	5	15	5	0f	1e	18	3	16	12	10	18	1d	0b	19	6	11	1b
page 98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 100	19	18	15	1c	10	1a	8	4	7	0c	18	17	9	17	1b	0c	15	9	0c	6	9	2	0c	18	5	6	1c	0a	7	5	1	1b
page 101	17	17	2	9	1a	10	18	7	1a	1b	5	16	1e	1	7	1c	15	0b	5	1b	1	1e	1	0a	0f	12	19	0c	1d	0a	0b	1e
page 102	7f	7f	7f	7f	7f	7f	7f	fb	7f	7f	7f	7f	7f	9d	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	ed	7f	7f	7f
page 103	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 104	0e	18	0f	2	0a	0b	1c	12	0e	13	1	15	19	14	0b	8	0	2	4	0e	18	6	0e	13	13	0	1c	12	9	7	15	1c
page 105	14	5	1	10	1	1c	0f	10	0c	14	1c	9	0b	12	9	12	10	0d	14	10	2	1e	0b	0d	0c	1c	1d	0f	0c	2	1b	5
page 106	16	2	11	3	10	15	15	0	10	5	0f	0f	0f	17	9	0e	7	6	13	4	18	0b	3	18	0	9	14	0	15	11	1e	9
page 107	0	1c	0e	1e	17	0e	1	3	18	0	1	15	12	14	2	1d	17	0	11	0a	7	13	1e	0	18	1c	2	1b	0	9	0	5
page 108	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 109	4	18	18	9	0	3	17	1e	2	7	0c	2	13	1a	1a	5	1c	18	4	1a	1c	1b	17	19	10	0	15	0f	4	4	1b	1c
page 110	14	2	9	1c	1e	1d	1b	1c	1b	0e	7	10	1	11	8	6	0d	14	0e	3	1e	6	11	19	15	0d	1b	2	10	3	0f	9
page 111	0d	0b	0a	1c	15	0a	0c	0b	3	1d	1a	1d	10	1	2	6	0e	9	1d	1b	0d	9	0b	0d	0d	3	19	16	1e	9	0d	18
page 112	7f	7f	7f	7f	7f	7f	7f	dc	7f	e1	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	d5	7f	7f	7f	7f	7f	7f	7f	7f	f5
page 113	7	11	1d	0b	16	0	17	1a	10	1b	12	18	0f	0b	8	11	17	1	14	1	13	16	1c	14	14	15	14	15	15	1a	0b	1d
page 114	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	84	7f	c7	7f	7f	7f	7f	7f	c0	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 115	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 116	11	17	17	1c	19	13	16	12	7	0	1c	3	0d	5	7	5	10	3	1a	12	4	6	16	1d	0c	0	7	11	0b	16	1	1e
page 117	16	13	1a	19	19	17	0c	17	0c	11	0	4	14	2	3	1b	1b	1e	18	9	12	10	2	4	13	0d	11	1c	2	4	1e	3
page 118	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
page 119	10	15	0d	9	5	11	15	10	0e	12	6	1c	1a	5	1a	15	1	17	19	9	6	0e	1	4	15	0b	0c	1c	18	9	1	1
page 120	1b	18	14	9	8	7	1	1c	9	0	9	3	0d	0a	5	0e	0c	0a	19	9	10	1b	1d	0b	1	1d	0f	6	3	7	9	4
page 121	1	0d	3	1d	1d	0d	1a	7	1a	5	9	10	17	0e	3	1	4	8	14	5	1c	14	9	11	1a	7	14	13	19	14	1	1a
page 122	e0	8e	a7	ab	ff	f2	bf	af	b6	f0	dd	ce	88	d4	97	fe	7f	aa	7f	7f	cc	c1	a2	e6	ad	a6	98	9f	87	7f	a5	d6
page 123	1b	13	1d	7	16	19	4	1c	1e	6	5	3	5	0a	18	6	7	15	19	0	18	15	9	0e	1	0d	0	17	10	5	6	12
page 124	0c	1b	17	0b	14	6	5	1	0b	7	10	1e	4	11	1a	12	10	17	0	1d	15	19	19	3	2	5	0f	1d	1c	0c	13	16
page 125	0f	2	0a	19	0	13	10	16	0b	1d	3	0c	9	11	6	0b	0d	1c	0	8	4	16	17	0	18	1b	0e	12	18	1c	6	15
page 126	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	85	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 127	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	83	7f	e9	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f

This page was intentionally left blank.