**CSED211 Homework #6**

**20210643 Kim HyunJune**

**1. Exercise 9.15 on page 915 with the following requests.**

풀이) Block size는 요청한 데이터 size와 헤더 size의 합을 구하고 8byte의 배수가 되도록 올림해서 구하면 되고, Blcok header의 경우 Blcok size의 hexadecimal 표현에서 LSB를 1로 바꾸어 주면 되므로, 답은 아래와 같이 나온다.

|  |  |  |
| --- | --- | --- |
| Request | Block size (decimal bytes) | Block header (hex) |
| malloc(3) | **8** | **0x09** |
| malloc(15) | **24** | **0x19** |
| malloc(25) | **32** | **0x21** |
| malloc(33) | **40** | **0x29** |

**2. Exercise 9.16 on page 915**

풀이) Allocated 및 Free 경우의 최소 사이즈를 구하여 더 큰 것이 minimum block size가 될 것이다. Allocated의 경우에는 header, footer, payload(최소 payload인 1byte)와 padding의 합으로 총 size가 계산되고, Free 부분이 Linked-List처럼 연결되는 Explicit Free list이므로 Free 경우에는 header와 footer, pred와 succ pointer가 들어가게 된다. 따라서 계산해보면 첫 줄은,

Allocated: 4(header)+4(footer)+1(minimum payload)+3(padding)=12이고,

Free: 4(header)+4(footer)+4(pred)+4(succ)=16이므로 16이 minimum block size이다.

같은 방법으로 둘째 줄은 Allocated: 4+1+3=8, Free: 4+4+4+4=16이므로 16이 되고,

3, 4줄은 double word alignment이므로 이에 맞추어 padding을 넣어주면,

셋째 줄은 4+4+1+7=16, 4+4+4+4=16, 넷째 줄은 4+1+3=8, 4+4+4+4=16이다.

따라서 답은 아래와 같이 나온다.

|  |  |  |  |
| --- | --- | --- | --- |
| Alignment | Allocated block | Free block | Minimum block size(bytes) |
| Single word | Header and footer | Header and footer | **16** |
| Single word | Header, but no footer | Header and footer | **16** |
| Double word | Header and footer | Header and footer | **16** |
| Double word | Header, but no footer | Header and footer | **16** |

**3. Exercise 9.19 on page 916**

**1. (a) In a buddy system, up to 50% of the space can be wasted due to internal fragmentation.**

(a)는 TRUE이다.

**(b) The first-fit memory allocation algorithm is slower than the best-fit algorithm (on average).**

First-fit memory allocation algorithm의 경우 처음부터 서치하여 사용 가능한 블록을 선택하지만, bet-fit algorithm의 경우 모든 사용 가능한 블록을 서치하기 때문에 평균적으로 best-fit이 느리다. 따라서 (b)는 FALSE.

**(c) Deallocation using boundary tags is fast only when the list of free blocks is ordered according to increasing memory addresses.**

Free block의 list가 address 순서가 아니더라도 LIFO 방식으로 유지만 된다면, 빠르게 boundary tag를 사용하는 deallocation을 할 수 있으므로, (c)는 FALSE.

**(d) The buddy system suffers from internal fragmentation, but not from external fragmentation.**

buddy system에서는 internal fragmentation뿐만 아니라 external fragmentation도 일어난다. 따라서 (d)는 FALSE.

따라서 답은 **(a)**이다.

**2. (a) Using the first-fit algorithm on a free list that is ordered according to decreasing block sizes results in low performance for allocations, but avoids external fragmentation.**

First-fit algorithm은 처음부터 서치하기 때문에 decreasing block size로 정렬되면 allocation이 수월하다. 따라서 (a)는 FALSE.

**(b) For the best-fit method, the list of free blocks should be ordered according to increasing memory addresses.**

Free block list를 정렬하는 것은 best-fit method가 아니라 first-fit method에서 탐색 시간을 줄일 수 있다. Best-fit에서 block이 정렬되었더라도 사용 가능한 블록을 모두 탐색하는 것은 똑같기 때문이다. 따라서 (b)는 FALSE.

**(c) The best-fit method chooses the largest free block into which the requested segment fits.**

best-fit method는 smallest free block을 선택한다. 따라서 (c)는 FALSE.

**(d) Using the first-fit algorithm on a free list that is ordered according to increasing block sizes is equivalent to using the best-fit algorithm.**

(d)는 TRUE이다.

따라서 답은 **(d)**이다.

**3. Mark&Sweep garbage collectors are called conservative if**

**(a) They coalesce freed memory only when a memory request cannot be satisfied.**

**(b) They treat everything that looks like a pointer as a pointer.**

**(c) They perform garbage collection only when they run out of memory.**

**(d) They do not free memory blocks forming a cyclic list.**

Mark&Sweep garbage colloector는 pointer처럼 보이는 모든 것들을 pointer로 다루면 conservative하다고 말한다. Pointer가 아닌 것들이 Pointer로 취급되면 unmark될 block이 mark되는 경우가 생긴다. 따라서 답은 **(b)**이다.

**4. Exercise 10.6 on page 950.**

File Descriptor 0, 1, 2는 이미 할당되어 있고, foo.txt와 bar.txt를 open할 때 fd1과 fd2에는 각각 3, 4가 리턴되어 들어가게 된다. Open 함수는 프로세스 내 열려 있지 않은 상태의 가장 작은 descriptor를 리턴하기 때문이다. 다음으로, close(fd2)가 실행되어 file descriptor 4는 정리가 된 이후, baz.txt를 Open하였으므로, 이때 4가 리턴되게 된다. 따라서 fd2의 값은 4이므로, fd2=4가 출력되게 된다. 따라서 답: **fd2 = 4**

**5. Exercise 10.9 on page 950**

< foo.txt 명령으로 Standard input에서 foo.txt로 input이 바뀌어야 하는데, 그러지 않아 descriptor 3의 file entry가 descriptor 0의 foo.txt entry와 대응되지 않아서 오류가 생겼다. 따라서 먼저 foo.txt를 열고, 해당 file descriptor를 Standard Input으로 redirect한 후, execve를 실행하여 fstatcheck의 input이 되어 들어가도록 하는 과정이 필요하다. 그러므로, Dup2(3,0);을 해당 위치에 적으면 이 문제가 해결되고 적절히 동작할 것이다. 따라서 답: **Dup2(3,0);**

**6. Enumerate the difference between process and thread concisely.**

1) process는 processor에서 실행 중인 프로그램을 뜻하고, 하나 이상의 thread로 scheduling을 진행한다. 또한 process는 자신만을 위한 memory를 할당 받게 되고, 해당 memory를 다른 process와 공유하지 않는다. 그러므로 특정 process에 exception이 발생하는 경우에 강제 종료되어도 큰 영향을 다른 process에 미치지 않는다.

2) 반면 thread는 process 안에서 실행되는 작업의 흐름을 말한다. Thread가 여러 개 있으면 multi thread이고, thread는 해당 thread를 다루는 stack 공간을 할당 받게 되고, 나머지 heap, code, data 등은 다른 thread와 공유한다. thread끼리의 공유를 위해 공유 영역을 접근할 수 있으므로 thread는 process보다 context switching이 빠른 모습을 나타낸다.

**7. Exercise 12.6 on page 1031**

**a)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable**  **Instance** | **Referenced by** | | |
| **Main thread?** | **Peer thread 0?** | **Peer thread 1?** |
| **ptr** | **Y** | **Y** | **Y** |
| **Cnt** | **N** | **Y** | **Y** |
| **i.m** | **Y** | **N** | **N** |
| **msgs.m** | **Y** | **Y** | **Y** |
| **myid.p0** | **N** | **Y** | **N** |
| **myid.p1** | **N** | **N** | **Y** |

**b)** dptr, cnt, msgs.m이 공유된다.

**8. Exercise 12.17 on page 1067**

**A.** main thread에서 exit으로 프로그램이 종료될 때, 다른 thread가 작동하기까지 대기하지 않고 바로 종료되므로, Sleep(1)이 실행되지 않고 바로 종료가 된다. 따라서 출력은 없다.

**B.** pthread\_exit을 exit 대신 사용하게 되면, 실행 중인 다른 thread가 종료될 때까지 대기하는 과정이 생긴다. 따라서 에러를 고칠 수 있을 것이다.