(a) Concept Questions

1. **Paging method**: Divide memory into fixed-size pages and frames. Use a page table to map pages to frames.
2. **TLB**: A fast cache for recently accessed page table entries to speed up address translation.
3. **Illegal page access**: OS detects via page table permissions; triggers a page fault if access is unauthorized.
4. **Segmentation vs. Paging**: Segmentation divides memory by logical units, can cause external fragmentation. Paging uses fixed-size units, no external fragmentation.

(b) Additional Questions

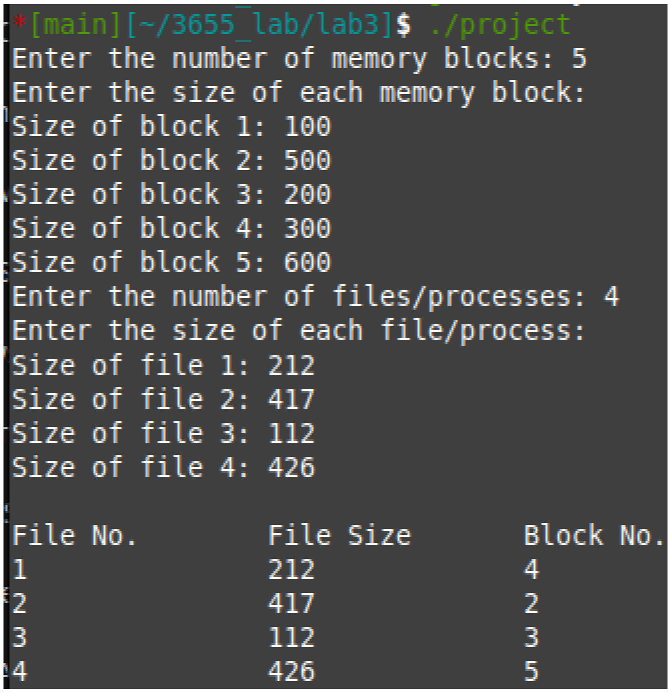
1. **Demand-paging vs. Swapping**: Demand paging loads pages on demand. Swapping moves entire processes in/out of memory.
2. **Extending TLB**: More entries can reduce misses but may slow TLB. Increasing page size can cause internal fragmentation.

Code:

```
#include <stdio.h>  // Include standard input/output header file

int main() {
   int nb, nf, i, j;  // Declare variables for number of blocks, number of files, and loop counters

   // Prompt the user to enter the number of memory blocks
   printf("Enter the number of memory blocks: ");
   scanf("%d", &nb);  // Read the number of memory blocks from user input

   int blockSize[nb], blockFlag[nb];  // Arrays to store block sizes and their allocation status

   // Initialize all blocks as free (blockFlag = 0)
   for(i = 0; i < nb; i++) {
      blockFlag[i] = 0;  // Set blockFlag[i] to 0 indicating block is free
   }

   // Prompt the user to enter the size of each memory block
   printf("Enter the size of each memory block:\n");
   for(i = 0; i < nb; i++) {
      printf("Size of block %d: ", i + 1);  // Display block number
      scanf("%d", &blockSize[i]);  // Read size of block i
   }
```

```
// Prompt the user to enter the number of files/processes
printf("Enter the number of files/processes: ");
scanf("%d", &nf);  // Read the number of files from user input

int fileSize[nf], allocation[nf];  // Arrays to store file sizes and allocation details

// Initialize all files as unallocated (allocation = -1)
for(i = 0; i < nf; i++) {
    allocation[i] = -1;  // Set allocation[i] to -1 indicating file is not allocated
}

// Prompt the user to enter the size of each file/process
printf("Enter the size of each file/process:\n");
for(i = 0; i < nf; i++) {
    printf("Size of file %d: ", i + 1);  // Display file number
    scanf("%d", &fileSize[i]);  // Read size of file i
}

// Implement the Best Fit algorithm for memory allocation
for(i = 0; i < nf; i++) {  // Loop through each file
    int bestIdx = -1;  // Initialize bestIdx to -1 (no suitable block found yet)
    for(j = 0; j < nb; j++) {  // Loop through each memory block
        // Check if block is free and can accommodate the file
        if(blockFlag[j] == 0 && blockSize[j] >= fileSize[i]) {
            // If it's the first suitable block or a better fit is found
            if(bestIdx == -1 || blockSize[j] < blockSize[bestIdx]) {
                bestIdx = j;  // Update bestIdx to current block index
            }
        }
    }
    // If a suitable block was found
    if(bestIdx != -1) {
        allocation[i] = bestIdx;  // Allocate block to file
        blockFlag[bestIdx] = 1;  // Mark block as occupied
    }
}

// Display the allocation results
printf("\nFile No.\tFile Size\tBlock No.\n");
for(i = 0; i < nf; i++) {
    printf("%d\t\t%d\t\t", i + 1, fileSize[i]);  // Display file number and size
    if(allocation[i] != -1) {
        printf("%d\n", allocation[i] + 1);  // Display allocated block number (1-based index)
        } else {
            printf("Not Allocated\n");  // Indicate file was not allocated
        }
    }

    return 0;  // End of program
}
```

Output Sample:



```
*[main][~/3655_lab/lab3]$ ./project
Enter the number of memory blocks: 5
Enter the size of each memory block:
Size of block 1: 100
Size of block 2: 500
Size of block 3: 200
Size of block 4: 300
Size of block 5: 600
Enter the number of files/processes: 4
Enter the size of each file/process:
Size of file 1: 212
Size of file 2: 417
Size of file 3: 112
Size of file 4: 426

File No.        File Size       Block No.
1               212             4
2               417             2
3               112             3
4               426             5
```