1  Bits (30 minutes, 20 points)

```c
#include <ctype.h>
#include <assert.h>
#include "word.h"

WERD W_make(const char *s) {
    unsigned long w=0;
    int i;
    assert (s!=NULL)
    for (i=0; s[i]; i++) {
        assert (i<12);
        assert (islower(s[i]));
        w = w | ((s[i]&0x1fL)<<5*i);
    }
    return w;
}

void W_get(WERD w, char *s) {
    int i;
    for (i=0; w; i++) {
        s[i]= (w&0x1fL)+('a'-'1L);
        w = w >> 5;
    }
    s[i]=0;
}
```

2  Explain a program (55 minutes, 35 points)

A. If key is already in the table, return 0. Otherwise, add key to the table and return 1.

B. Must be called with exactly one argument that is a decimal integer between 1 and MAX, otherwise silently exits with code 1. Then, chooses k random numbers between 0 and 5478 (inclusive), and prints out how many distinct numbers there were.

C. With few buckets in the hash table, the average linked-list size is large, which takes time to traverse.

D. With many more buckets than entries in the hash table, almost all of the buckets are empty, and the spatial and temporal locality of accessing `table[b]` is bad: there's a cache miss at almost every access of `table[b]`.

E. N, table: bss;      num, b,p: registers (or stack).

F. 100

G. 5479 (answers between 5470 and 5478 are acceptable)
3. **Assembly language** (35 minutes, 25 points)

A.

```
.equ key, 0
.equ next, 8 # we didn't take points off if you said 4 instead of 8

.section .bss
table: .skip 800000000
```

B.

```
movq table(%r12,8), %rbp
loop:
  cmpq $0, %rbp
  je done
  cmpl %edi, key(%rbp)
  jne else
  movl $0, %eax
  popq %rbx
  popq %rbp
  popq %r12
  ret
else:
  movq next(%rbp), %rbp
  jmp loop
done:
```
4 Processes (40 minutes, 20 points)

A. (worth 10 points)
child: hello
child: goodbye

B. No.

C.
1. child’s execvp fails, therefore returns
2. child reaches line 22
3. child finishes if-statement, reaches line 23
4. in child line 23, close is redundant, no effect
5. in child line 24, fdopen fails because p[0] is already closed in child, f is set to NULL
6. in child line 26, fgets when f==NULL returns NULL loop terminates
7. child reaches line 29, child exits
8. only now, no more threads have writing end of pipe open, so the parent’s fgets() returns NULL
9. parent executes normally, without having printed anything.

D. insert after line 21 either one of the following:

/* option 1*/ fprintf(stderr, "cannot execute /bin/echo
\n"); exit(1);
/* option 2*/ perror("execvp failed"); exit(1);

If you used access instead, this doesn’t work as reliably, but we didn’t take off points.

Remark: “process table is full” in the question is actually not a failure mode of execvp, it would cause fork to fail. I didn’t intend to make the question misleading in that way. My bad.