C programming in Linux

CS 342 – Operating Systems
Spring 2004

Ibrahim Korpeoglu
Bilkent University

- Editing and Compiling
- Makefile
- Debugging
- Revision control - RCS

Editing

- There are various editors
  - emacs
  - vi
  - kwrite
  - pico
  - joe
  - ….

Compiling

- Use gcc
- We can compile at the command line
- gcc may have various options
  - These options are important to know:
    - -o (with this we can specify the name of the executable file)
    - -g (force the compiler to produce debugging info in the executable so that we can debug the executable with a debugger like gdb or xxgdb)
    - -Wall (for the compiler to output all the warnings. It is important to remove all the warnings as well. You should compile your programs always with this option.)

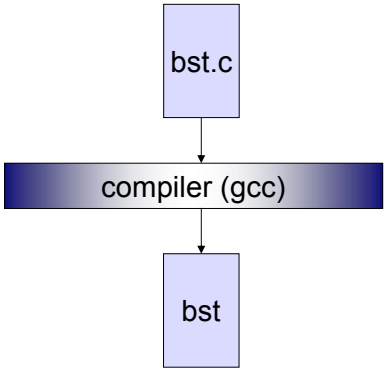
Compiling - debugging

- In order to have the compiler generate debugging instructions into our executable file, we have to invoke the compiler with
  –g option
  - gcc –g bstsort.c
- If we have a segmentation fault while running our program, the memory state and state of CPU registers are dumped into a file called “core…” in our directory, provided that the limit of the core file-size is not zero.
- You can learn the limits by “ulimit –a” command
- You can set the limit of core file size with “ulimit –c <limit>”, for example “ulimit –c 10000000” will set the limit to 10 million bytes. In this case the core file that is created when a segmentation fault happens can not be bigger than 10 million bytes.

Compiling - debugging

- We can use the core file to find out where we got a segmentation fault (at which line of our .c code)
- For this, we can use the gdb debugger.
- Just type:
  - gdb <programname> <corefilename>
    - For example: gdb bstsort core
      or gdb bstsort core.9657
      (here 9657 is the process ID number of the process that caused the core file to be created upon segmentation fault)
  - It will show you the line where the segmentation fault has occurred.
  - Try also with “list” command of gdb at the command prompt of the gdb. You get the gdb command prompt when you start gdb.

Compiling Example:



gcc –o bst –g –Wall bst.c

Compiling with Makefile

- We can use the “make” utility to compile our programs.
- Very useful for large projects (Consisting of 10s, 100s, 1000s of files)
- The make utility reads the commands (rules) to compile a program from a file called “Makefile” that is usually located in the same directory as the source files.

Makefile Example

```
homework: bst.c
	gcc -g -o bst bst.c
<TAB>
clean:
	rm -fr *~ a.out bst
<TAB>
```

Makefile

- At command prompt, we just type “make” and invoke the make utility in this way.
- The make utility opens and reads the file called Makefile located in the same directory.
- Note that we use of TABs instead of spaces for indenting the lines in a Makefile.

RCS

- RCS: revision control system
- Can be used to keep track of versions of your program(s).
- You can store a version of your C code in RCS system and then later retrieve it when you want to do some modifications.
- You can store as many revisions as you wish.

Some Important RCS commands

- "ci –l filename.c"
  - Stores the revision in the RCS database
  - A copy of the file stays in the directory, so that the user can continue working on the file.
- "co –l filename.c"
  - Retrieves the file from the database and locks it, so that the retrieved file can edited and modified.
- rlog
  - List all the revisions
- ident

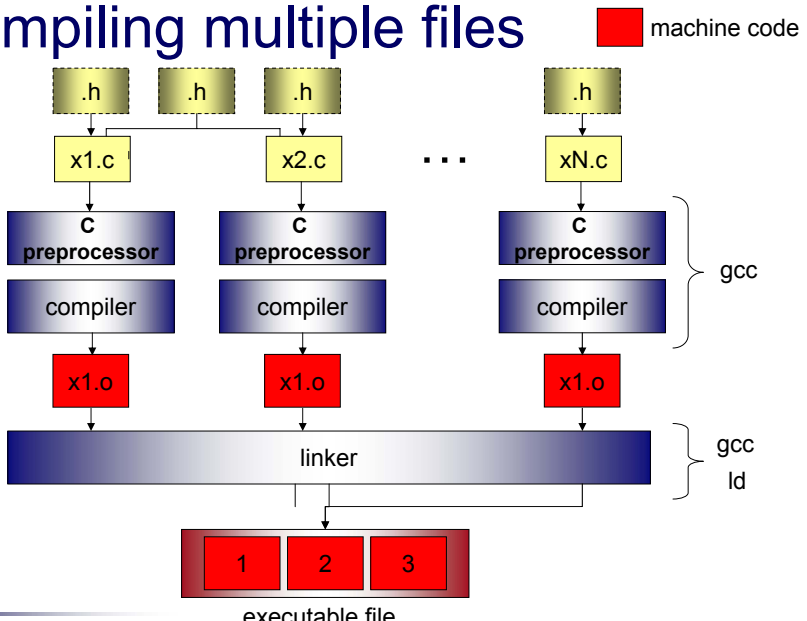
Some important RCS commands

- Ident
  - Give information about the files, such as author, revision number, etc.
  - For this information to appear, the file should have the following string somewhere at the top:
    /* Id */
  - When you check in the file, this string is replaced with the revision number and other information.
  - When you ckeck out the file, and look to the inside of the file, you will see the "Id" replaced with the revision number and other information about the file.

RCS

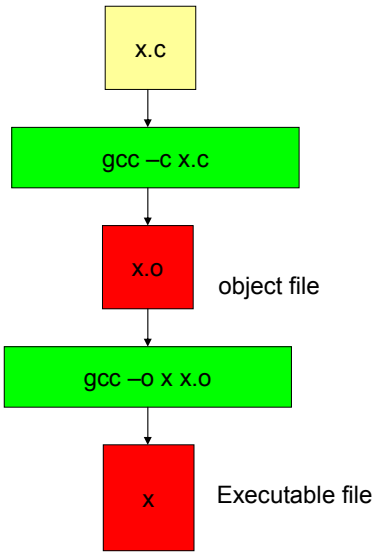
- You can learn more about RCS by typing "man rcsintro"
- There are other related RCS commands like:
  - rcs
  - rcsmerge
  - rcsdiff (show the different between two revisions). You have to specify the revision number. Read the mane page for "rcsdiff".

Compiling multiple files



Object file

- -c option of gcc causes only the object file to be produced.
- Object file has the machine code. Besides some other information, of the program. The machine code is translated from the C source file by the compiler.
- gcc –c x.c
- gcc x.o –o x



Analysis of our homework program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node {
  struct node *left;
  struct node *right;
  int         count;
  char        *word;
};
```

```
void
insert_bst (struct node *r, struct node *n)
{
  int ret;

  ret = strcmp(n->word, r->word);
  if (ret == 0)
    r->count++;
  else if (ret < 0) {
    if (r->left == NULL)
      r->left = n;
    else
      insert_bst (r->left, n);
  }
  else if (ret > 0) {
    if (r->right == NULL)
      r->right = n;
    else
      insert_bst (r->right, n);
  }
}
```

Point13

Point14

Point15

Point16

```
void print_bst(struct node *r)
{
  int i;

  if (r != NULL) {
    print_bst (r->left);
    for (i=0; i < r->count; ++i)
      printf("%s\n", r->word);
    print_bst (r->right);
  }
}

void free_bst (struct node *r)
{
  if (r != NULL) {
    free_bst (r->left);
    free_bst (r->right);
    free(r->word);
    free(r);
  }
}
```



```
int
main(int argc, char *argv[])
{
  FILE * fp;
  struct node *root;
  struct node *newnode;
  char   buffer[1024];

  if (argc != 2) {
    printf("wrong number of arguments\n");
    exit(1);
  }

  if ( (fp = fopen(argv[1], "r")) == NULL) {
    printf ("can not open the file \n");
    exit(1);
  }

  root = NULL; /* initialize the root pointer */
```

Point 0

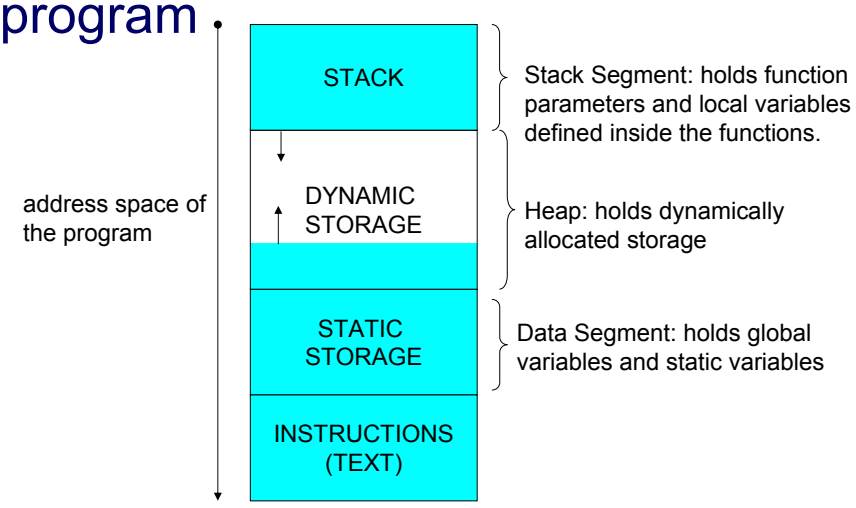
Point 1

Point 2

```
if (fscanf(fp, "%s", buffer) == 1) {          Point3
   newnode = (struct node *) malloc           Point4
(sizeof(struct node));
   newnode->left = NULL;
   newnode->right= NULL;
   newnode->count = 1;                        Point5
   newnode->word = (char *) malloc
(strlen(buffer) + 1);                         Point6
   strcpy(newnode->word, buffer);             Point7

   root = newnode;                            Point8
 }
```

```
while ( fscanf(fp, "%s", buffer) == 1 ) {     Point9
   newnode = (struct node *) malloc
(sizeof(struct node));                        Point10
   newnode->left = NULL;
   newnode->right= NULL;
   newnode->count = 1;
   newnode->word = (char *) malloc
(strlen(buffer) + 1);                         Point11

   strcpy(newnode->word, buffer);             Point12

   insert_bst(root, newnode);                 Point17
 }

 print_bst(root);

 free_bst(root);
 root = NULL;
 exit(0);
} /* end of main */
```


General memory layout of a program

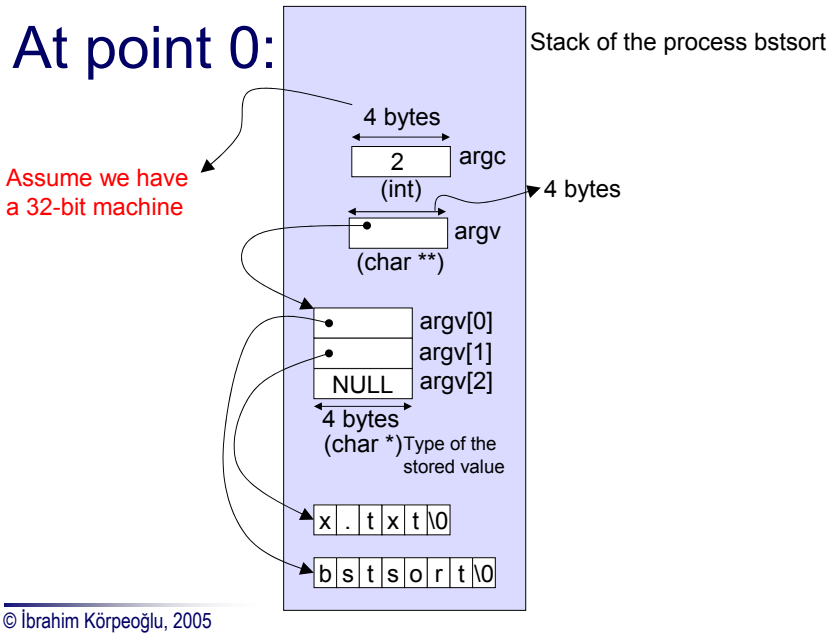


Running our program

- Assume our program is run as:
  - bstsort x.txt
- Assume the input file x.txt has the following words in it:

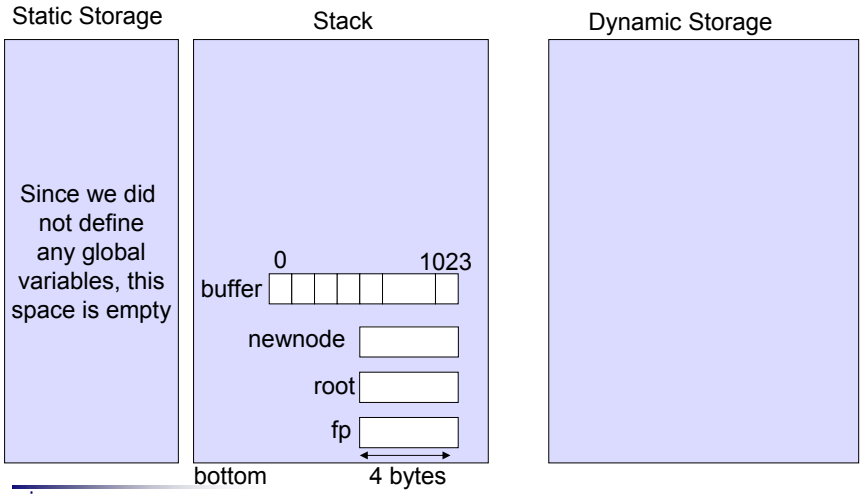
```
ali
veli
.....
```


At point 0:

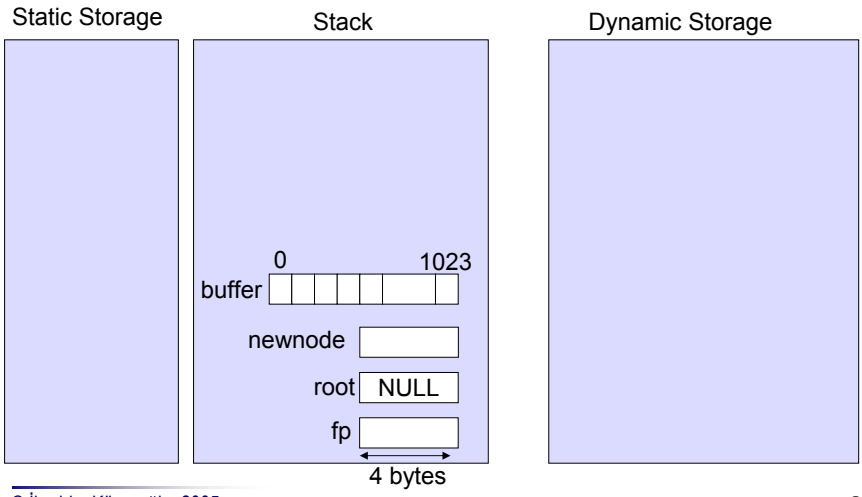


- When our program is run as the following:
  bstsort x.txt
- At point 0: our program will have two parameters argc, and argv filled up with values that we were obtained from the command shell while starting up the program.
- These variables (argc and argv) will stay in the bottom of the stack until main() returns, i.e. until the program terminates. We will not show them in later slides, but you should just know that they will be there during the lifetime of the main function. We do not show them just because of space restrictions on our slides.

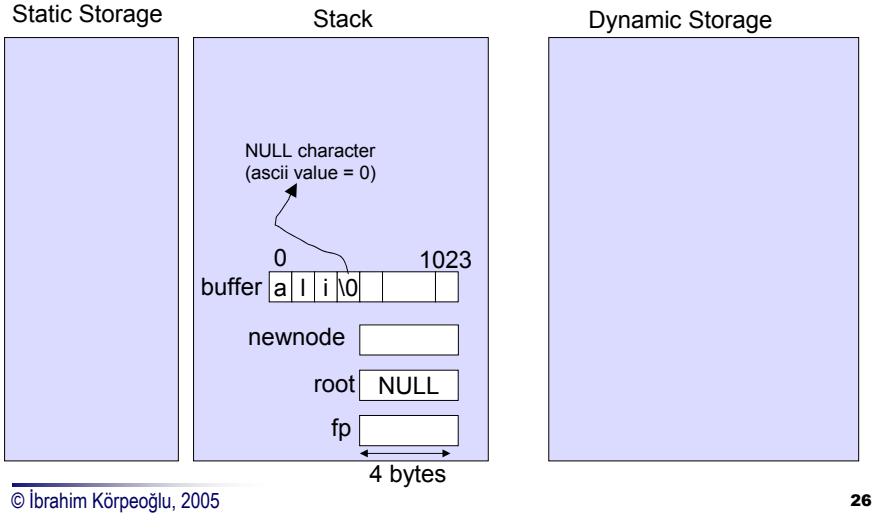
At point 1: state of various storage segments in memory (i.e. state of segments of the program)



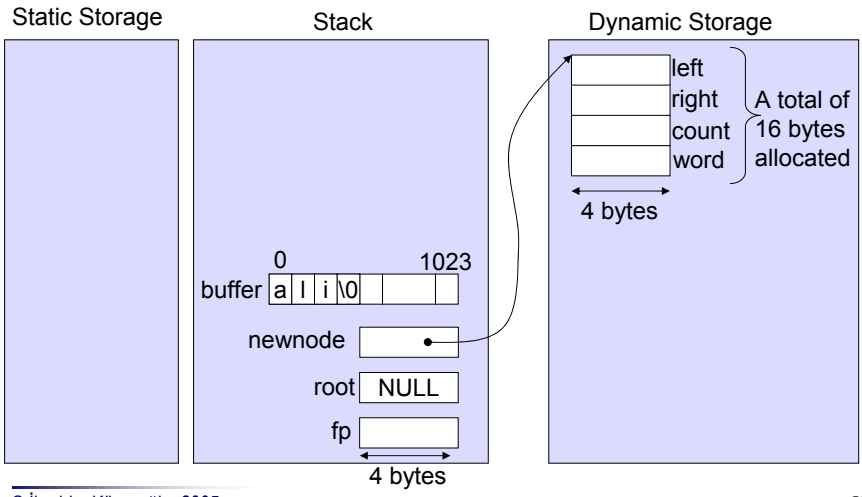
At point 2:



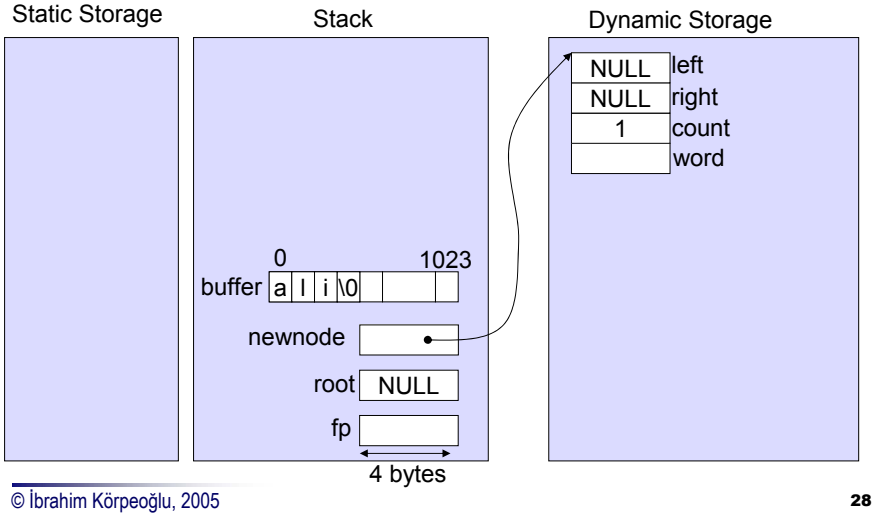
At point 3:



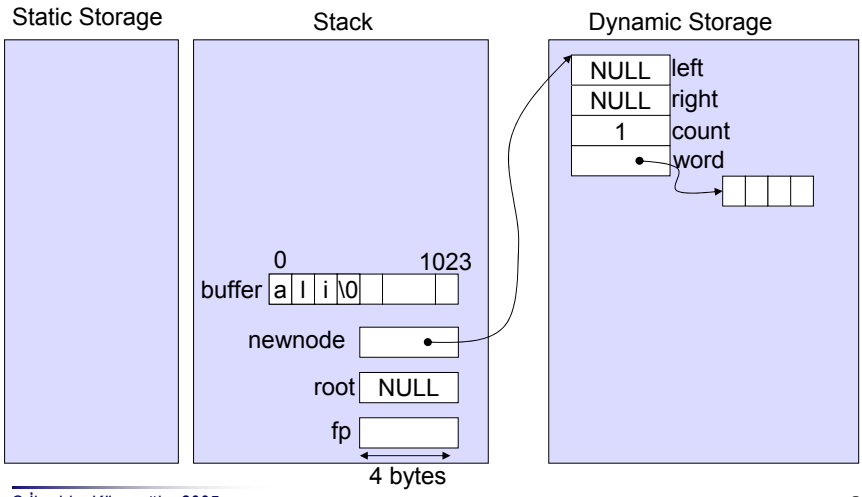
At point 4:



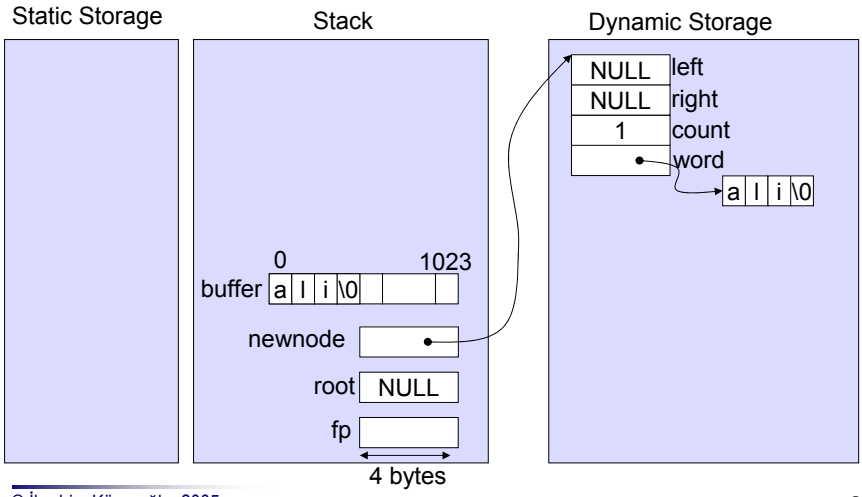
At point 5:



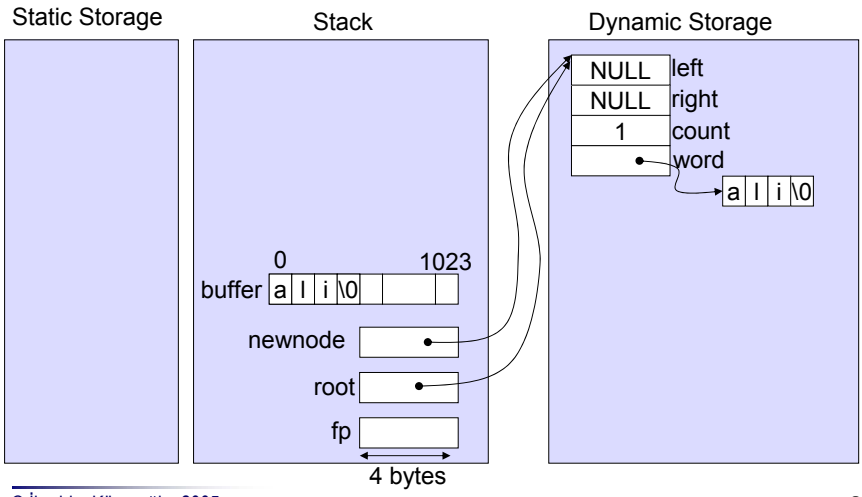
At point 6:



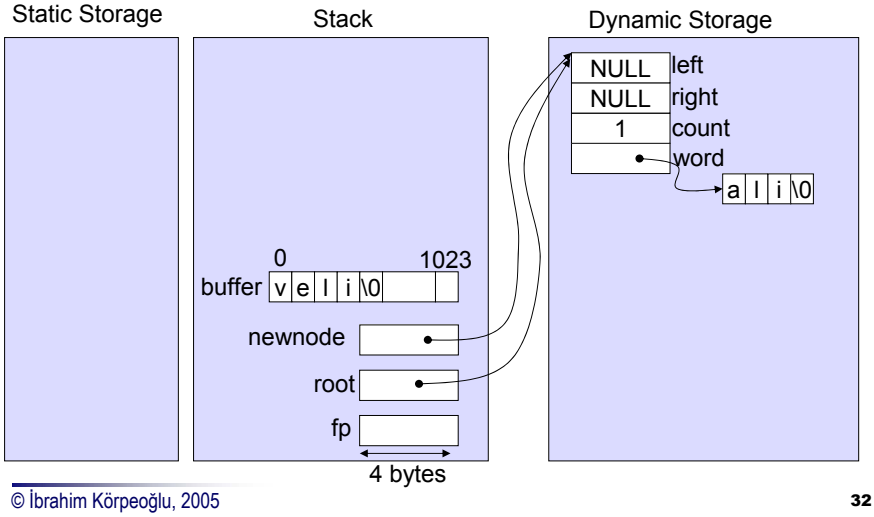
At point 7:



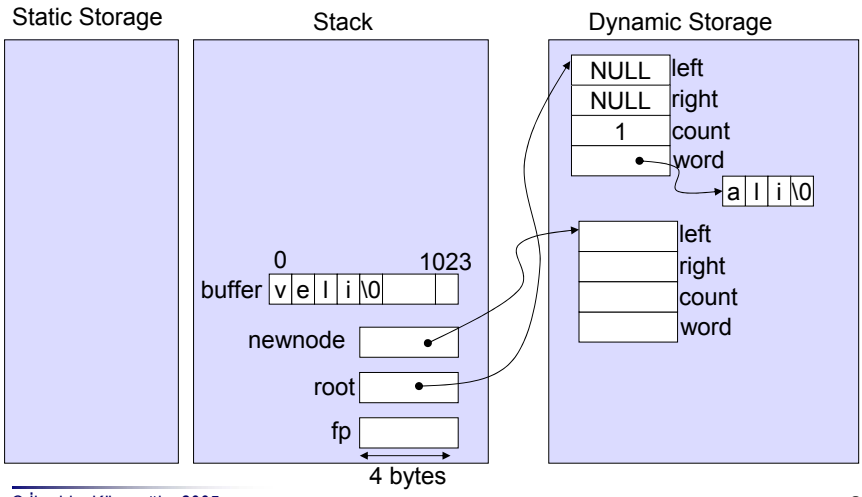
At point 8:



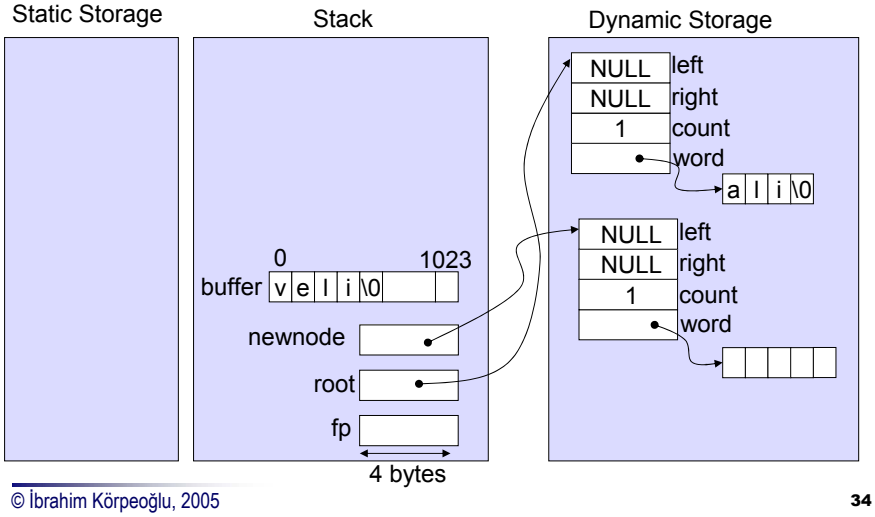
At point 9:



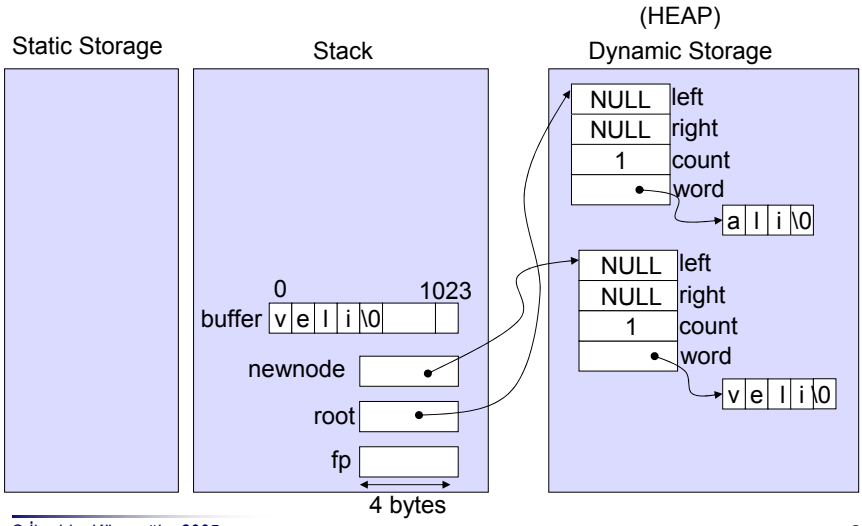
At point 10:



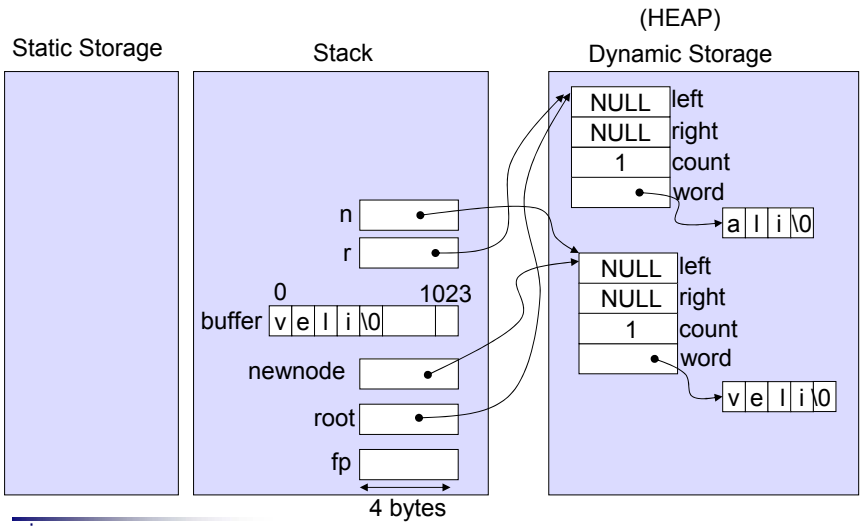
At point 11:



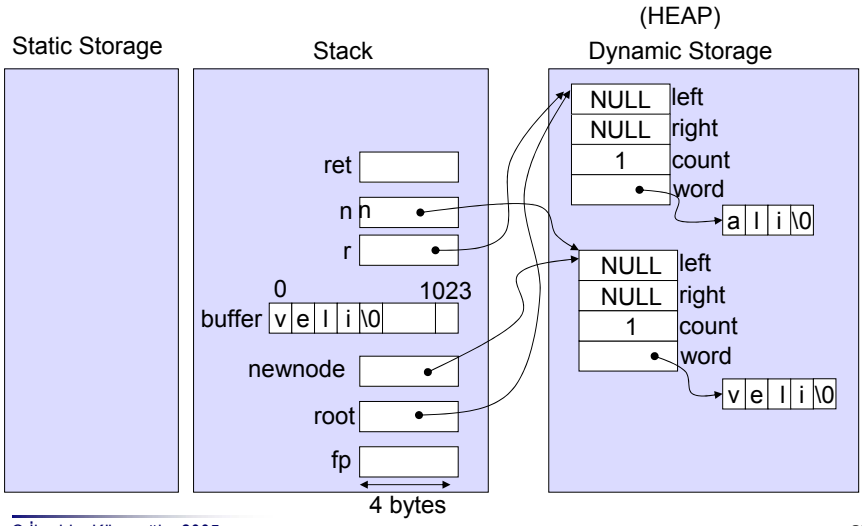
At point 12:



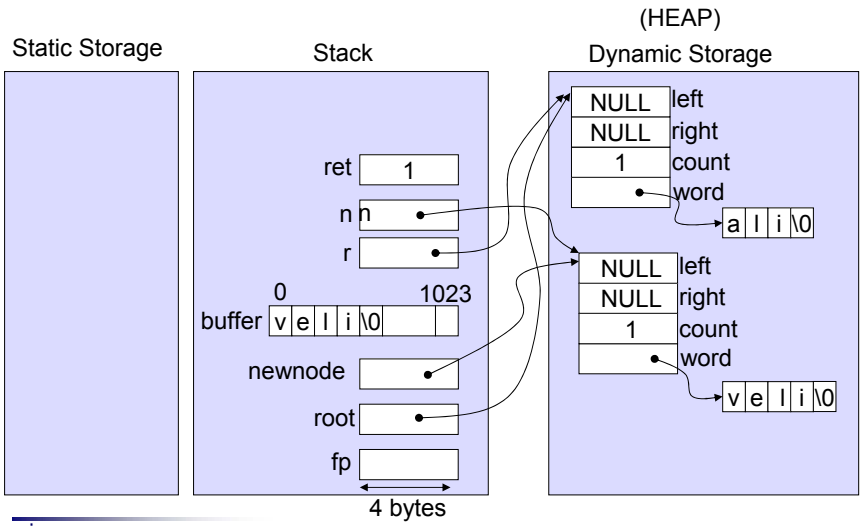
At point 13:



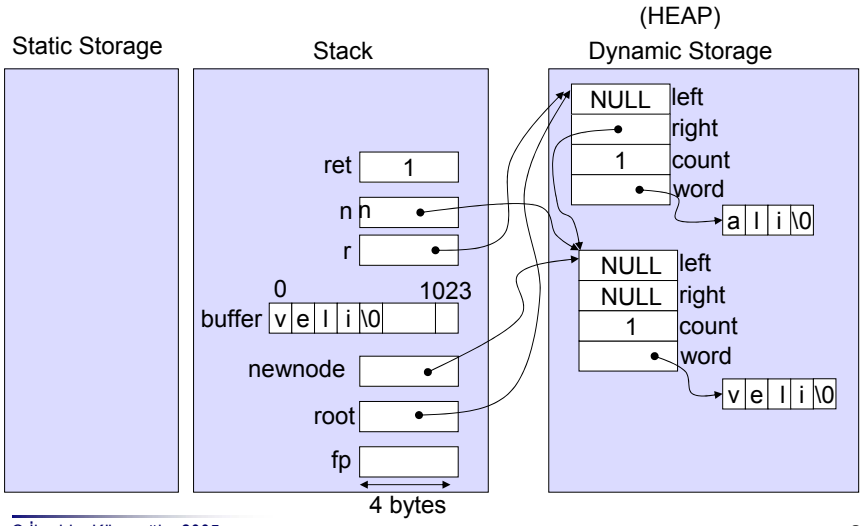
At point 14:



At point 15:



At point 16:



At point 16: