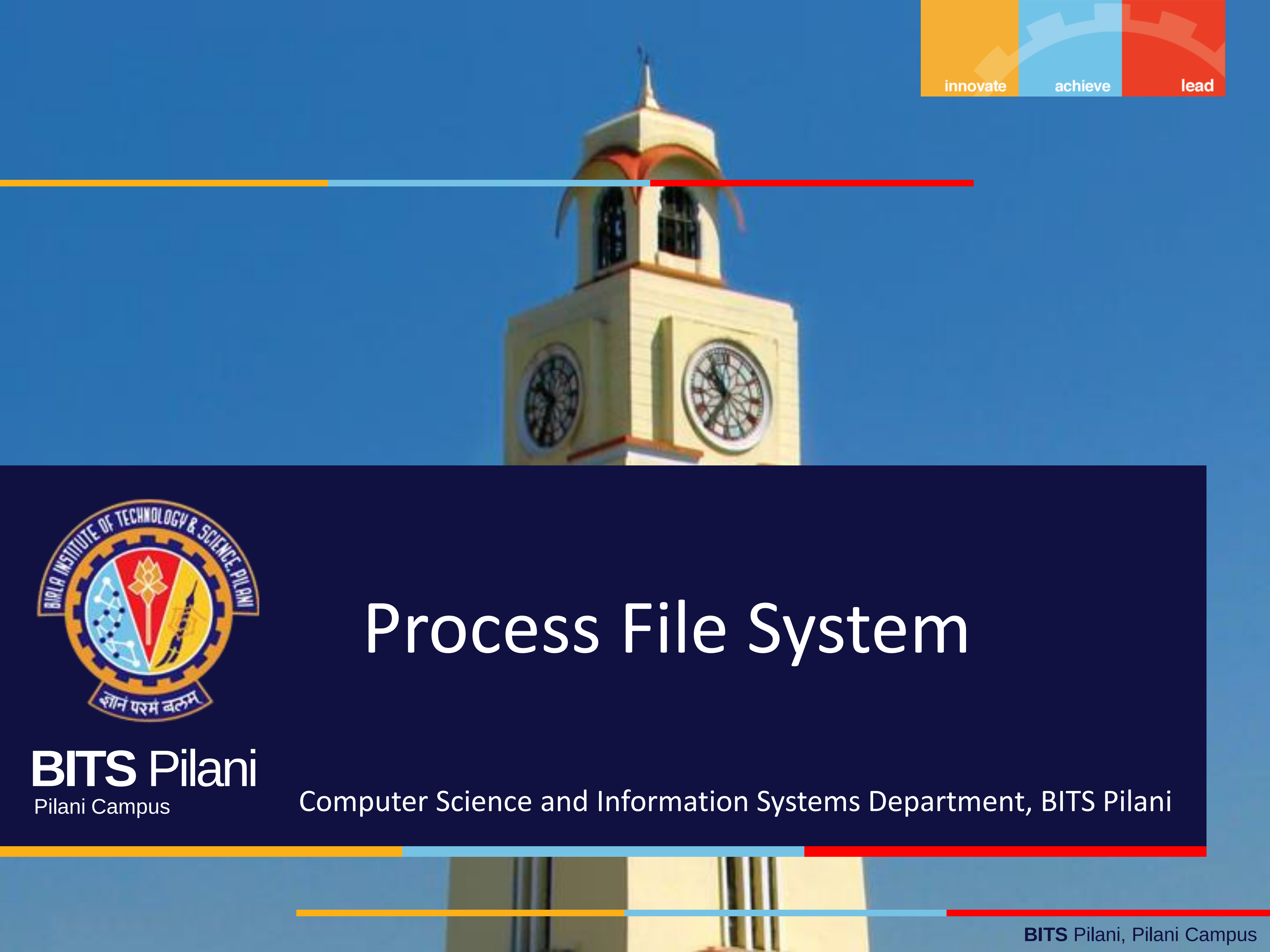
innovate achieve lead



Process File System

BITS Pilani
Pilani Campus

Computer Science and Information Systems Department, BITS Pilani

Today’s Agenda

- Unix System Calls
  - Process Creation
  - Process Execution

Exec() System Call

exec() system call

- exec system call is used to replace the old file or program from the process with a new file or program.
- The new program is loaded into the same process space.
- The current process is just turned into a new process and hence the process id PID is not changed.
- PID of the process is not changed but the data, code, stack, heap, etc. of the process are changed and are replaced with those of newly loaded process.
- The new process is executed from the entry point.

Exec Family

S.No.	**A**	**B**	**Name**	**Meaning**
1	l		***execl***	Execute file with arguments explicitly in call
2	v		***execv***	Execute file with argument vector
3	l	p	***execlp***	Execute file with arguments explicitly in call and PATH search
4	v	p	***execvp***	Execute file with argument vector and PATH search
5	l	e	***execle***	Execute file with argument list and manually passed environment pointer
6	v	e	***execve***	Execute file with argument vector and manually passed environment pointer

Exec () System Call

- int **execl**(const char *path, const char *arg, ..., NULL);
- int **execlp**(const char *file, const char *arg, ..., NULL);
- int **execv**(const char *path, char *const argv[]);
- int **execvp**(const char *file, char *const argv[]);
- int **execle**(const char *path, const char *arg, ..., NULL, char * const envp[]);
- int **execve**(const char *file, char *const argv[], char *const envp[]);

execl() System Function:

```
# include <unistd.h>
int main(void) {
  char *binaryPath = "/bin/ls";
  char *arg1 = "-l";
  char *arg2 = "/home";
  execl(binaryPath, binaryPath, arg1, arg2, NULL);
  return 0;}
```

In execl() system function takes the path of the executable binary file (i.e. **/bin/ls**) as the first and second argument. Then, the arguments (i.e. **-l**, **/home**) that you want to pass to the executable followed by **NULL**. Then execl() system function runs the command and prints the output. If any error occurs, then execl() returns -1. Otherwise, it returns nothing.

execv() System Function:

```
#include <unistd.h>

int main(void) {
  char *binaryPath = "/bin/ls";
  char *args[] = {binaryPath, "-lh", "/home", NULL};
  execv(binaryPath, args);
  return 0;
}
```

In execl() function, the parameters of the executable file is passed to the function as different arguments. With execv(), you can pass all the parameters in a NULL terminated array **argv**. The first element of the array should be the path of the executable file. Otherwise, execv() function works just as execl() function.

execvp() System Function:

```
#include <unistd.h>

int main(void) {
  char *programName = "ls";
  char *args[] = {programName, "-l", "/home", NULL};
  execvp(programName, args);
   return 0;
}
```

Works the same way as execv() system function. But, the PATH environment variable is used. So, the full path of the executable file is not required just as in execlp().

execle() System Function:

```
#include <unistd.h>

int main(void) {
   char *binaryPath = "/bin/bash";
   char *arg1 = "-c";
   char *arg2 = "echo "Visit $HOSTNAME:$PORT from your browser."";
   char *const env[] = {"HOSTNAME=www.bits.com", "PORT=8080", NULL};
   execle(binaryPath, binaryPath,arg1, arg2, NULL, env);
   return 0;
}
```

Works just like execl() but you can provide your own environment variables along with it. The environment variables are passed as an array **envp**. The last element of the **envp** array should be NULL. All the other elements contain the key-value pairs as string.

execve() System Function:

```
#include <unistd.h>

int main(void) {
   char *binaryPath = "/bin/bash";
   char *const args[] = {binaryPath, "-c", "echo "Visit $HOSTNAME:$PORT
    from your browser."", NULL};
   char *const env[] = {"HOSTNAME=www.bits.com", "PORT=8080", NULL};
   execve(binaryPath, args, env);
   return 0;
}
```

Just like execle() you can provide your own environment variables along with execve(). You can also pass arguments as arrays as you did in execv().

execl() system Call

- It execute file with arguments explicitly in call.
- Syntax:

```
    int execl (
        const char *path, /* Complete Program pathname */
        const char *arg0, /* First Argument(filename) */
        const char *arg1, /* Second Argument(optional) */
        …                 /* Remaining Arguments (if any) */
        (char *) NULL    /* Arg list terminator */
);

/* Returns -1 on error (sets errno) */
```

execl() system Call

- After the call to execl() the context of the process is overwritten.
- Previous code is replaced by the code/instructions of the executable in ‘path’.
- User data is also replaced with the data of the program in ‘path’ thereby reinitializing the stack.
- And the new program begins to execute from its main function.
- New program accesses the arguments of new program which are mentioned in execl() through its ‘argc’ and ‘argv’ arguments of the main function.
- Environment pointed to by ‘environ’ is also passed to the new program.

Return of execl() system call

- Recall that the return address of any function is saved in the stack.
- The return address is popped from the stack while a function returns.
- But here the stack is reinitialized with the data of the new program and the old program’s data is lost.
- So there is no way to pop the return address and hence there is no way to return from execl() call if the call is successful.

execl(): example to invoke user executable

sum.c

```
int main(int argc,char *argv[])
{
int sum=0;
int i;
if(argc != 4)
{
printf("invalid argument\n");
    exit(0);}
for(i=0;i<argc;i++)
    sum = sum + atoi(argv[i]);
printf("sum = %d\n",sum);
}
```

TEST.C

```
int main()
{
execl("./sum","sum","100","200",
"300",(char *)NULL);
printf("execl call unsuccessful\n");
}
```

OUTPUT:-

600

execl(): example to invoke UNIX commands

```
# include<stdio.h>
# include<unistd.h>
int main(int argc, char ** argv){

printf("Hello World!");
execl("/bin/echo","echo","Print","from","execl",(char *)NULL);
return 0;
}
```

```
Output:- Print from execl
```

In the above program "Hello World!" is not printed

Reason:

Printf() function in C does not immediately prints the data on stdout but it buffers it till the next printf() statement or program exit.

Other exec system calls

- The other exec calls are very similar to execl(). They provide the following three features that are not available in execl().
  - Arguments can be put into a vector/array instead of explicitly listing them in the exec call. This feature is useful if the arguments are not known at compile time.
  - Searches an executable using the value of the PATH environment variable. When this feature is used we don't have to specify the complete path in the exec call.
  - Manually passing an explicit environment pointer instead of automatically using *environ*.

```
int execv (
const char *path, /* Program pathname */
char* const argv[] /* Argument vector */
);

int execvp (
const char *file,  /* Program filename */
char* const argv[] /* Argument vector */
);

int execve (
const char *path,  /* Program pathname */
char *const argv[], /* Argument vector */
char *const envv[]  /* Environment vector */
);
```

```
int execlp (
const char *file, /* Program filename */
const char *arg0, /*First Argument(filename) */
const char *arg1,
...
(char *) NULL     /* Arg list terminator */
);

int execle (
const char *path, /* Program pathname */
const char *arg0, /*First Argument(filename) */
const char *arg1,
...
(char *) NULL,     /* Arg list terminator */
char *const envv[]  /* Environment vector */
);
```

Fork() and wait() system call

```
int main(void)
{
int pid;
int status;
printf("Hello World!\n");
pid = fork( );
if (pid == -1) /* check for error in fork */
{
perror("fork failed");
exit(1);
}
if (pid == 0)
printf("I am the child process. %d\n",getpid());
else
{
wait(&status); /* parent waits for child to finish */
printf("Child Process with pid = %d completed with a status
   %d\n",pid,status);
printf("I am the parent process.%d\n",getpid());
}
```

output

```
Hello World!

I am the child process. 1928

Child Process with pid = 1928 completed with a status 7424

I am the parent process.1927
```

Any Queries?